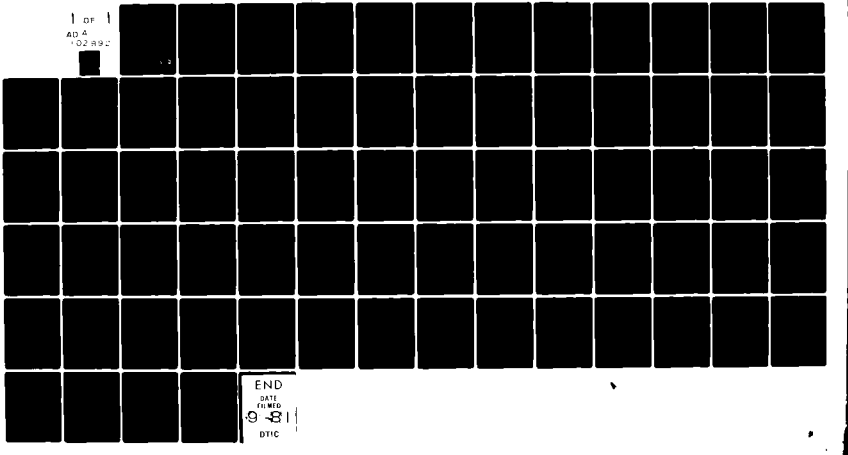
AD-A102 992 INSTITUTE FOR SOFTWARE ENGINEERING PALO ALTO CA F/G 9/2
QUEUEING NETWORK ANALYSIS IN SOFTWARE PHYSICS,(U)
APR 79 L M TRAISTER N00014-78-C-0768
UNCLASSIFIED NL



LEVEL II

QUEUEING NETWORK ANALYSIS IN SOFTWARE PHYSICS

by

L. M. Traister

April, 1979

Institute for Software Engineering
P.O. Box 637, Palo Alto, CA 94303

SECTION I

ABSTRACT

QUEUEING NETWORK ANALYSIS IN SOFTWARE PHYSICS

L. M. Traister
April, 1979

Queueing network models have achieved a prominent place in the modelling of computer system performance. Recent formulations of operational methods by Buzen and others have greatly facilitated their practical use. On the other hand, Kolence's software physics has addressed the problem of appropriate metrics for both performance and capacity of processors and systems and for the workloads with which they interact.

This paper integrates the methods of operational queueing network analysis into an extension of software physics. A significant feature of this approach is that parameters of the equipment and of the workload are not confounded in the analysis but are kept distinct, finally combining in the model computations proper. The objectives, principles and assumptions are stated and the basic quantities are defined. The fundamental operational laws and certain of the algorithms are derived and illustrated with examples.

This work was supported in part by the Office of Naval Research under Contract number N00014-78-C-0768

SECTION II

BACKGROUND AND OBJECTIVES

1.0 THE BACKGROUND OF OPERATIONAL ANALYSIS

Queueing network analysis is a means of arriving at the quantities that describe the performance of systems of limited resources. It provides understanding of how these systems react to the demands made upon them. Until recently, the queueing analysis methodology for computing systems has been more or less a direct adaptation of that for the traditional applications in that the performance quantities of response time, waiting time, server utilizations and so on are described by statistical distributions resulting from the statistical descriptions of the "customer" arrival rates and server service rates. Frequently this stochastic approach leads to formulations of much complexity, often with little hope of thorough solution. Moreover the focus has been observed to be more on the pursuit of nice, closed mathematical solutions, with less regard to providing a reasonably accurate assessment for the practical situation. As Newell [NEWE71] has commented, the situation has nearly become one of "solutions in search of a problem."

More recently, however, there has been work which maintains touch with the practical situation both in its formulation of relationships by insisting on *testability* and in its use of quantities which must be measurable in an implemented system. Buzen, in particular has argued for and developed such an approach and named it *Operational Analysis.* He has through its development created a useful approach to performance analysis which is more readily comprehended and put into practice.

2.0 THE BACKGROUND OF SOFTWARE PHYSICS

Kolence [KOLE76] has provided us with an approach to quantification and measurement of computer system attributes that is intimately connected with performance. These are the concepts and metrics

for software work, software and hardware execution times and the derived quantity of software power. The demand that a specific function places on equipment is measured in terms of work requirement; the capacity of the equipment to respond is measured in terms of power, that is, work divided by time. In addition to all this he has provided a hierarchical structure for system architecture that permits natural decompositions. These structural properties are explored in some detail in [KOVA79]. What is of importance for us here is that in using the software physics work demand and equipment power quantities rather than the more traditional quantities (in queueing analysis) of service time and visit ratios, we are able to keep properties of the workload (work demand and distribution) and properties of the equipment (software power) distinct in expression, so that the impact of each is separately visible and each is separately manipulable.

3.0 THE OBJECTIVE OF THIS EFFORT

What we have set out to realize in this current work is the uniting of the operational approach to computing systems performance analysis with the structural concepts and metrics of software physics. By doing so, we hope at least to establish a basis for a natural, engineering approach to operational queueing analysis for computing systems. Some of the benefits of this approach will be encountered in this paper, such as the isolation of workload and equipment properties discussed above or the simple way in which work distribution can be specified. In other cases we observe that just the translation of conventional forms into software physics notation can be revealing in isolating the parameters which affect performance. But we happily expect that this is just a beginning; that having put these formulations to use over extended periods, we will gain better insight not only into the systems that they describe but into the extension of the theory itself, thus giving an independent life to this mode of formulation and perception.

SECTION III

REQUIREMENTS, PRINCIPLES AND ASSUMPTIONS

3.0 INTRODUCTION

We discuss here some of the requirements for a methodology for the modelling and validation of computing system performance in which theory and practical application can be integrated. Furthermore, we state the basic principles and assumptions which support a straightforward and tractable development of the integrated methodology.

3.1 REQUIREMENTS - MODELLING AND VALIDATION

Briefly stated, our objective is to provide mathematical entities which characterize the performance of computer systems, that is, a mathematical model. A fundamental requirement to our development is that all hypotheses be capable of verification on the real system being modelled, a concept which Buzen calls *operational testability*. Unfortunately, the traditionally invoked methodology of stochastic modelling does not meet our fundamental requirement. This is because the basic assertion there is that the probability distributions of stochastic processes govern or characterize the behavior of real systems, an assertion that cannot be proved by measurement. Furthermore, even though stochastic methods are based on probabalistic considerations, their use describes only the variability (in the sense of uncertainty) of the quantities which are the parameters of the model, the process tells us nothing about the future values of those same quantities. Thus, the problem of parameter value prediction, as Buzen points out [BUZE77b] is the same for both operational and stochastic methods and here one might make use of probabalistic methods among others. However, this is an issue separate from that of which type of model to employ.

Concerning the parameterization of the models, we express a preference for formulations developed around those same quantities which we already use to characterize the workload or the capability of processors in a more general context. This naturally leads us to a choice of formulations based on work demands made on processors by jobs or transactions (which we will term *interactions*). The capability of the processors to service those demands will be described in terms of work performed per unit time or processor power. These quantities are in fact those of established software physics, which indeed characterizes workloads, processors and many of their interactions. It furthermore has an experimental basis providing methodology for measurement of many of these same quantities.

In the subsequent development, then, we will seek to express parameters of our models only in terms of established software physics entities or their simple extensions. In order to facilitate clear analysis, we wish to keep distinct those quantities which describe the workload and those which describe the equipment. Note that the quantities of conventional queueing analysis do not generally preserve this distinction. For example, the conventional "service time" quantity as in regard to, say, a disk direct access processor, derives from the action of a workload requirement (block length) applied to an equipment capability (access time and read/write rate). The quantities of software physics also carry with them clear specification for their measurement in a computing system context. These include the quantities that are the model parameters and those that describe performance.

3.2 PRINCIPLES AND ASSUMPTIONS

Although we will present these in context as needed, it is useful to indicate at once some of the basic principles and assumptions which will shape our operational methodology.

3.2.1 Flow Balance

This conservation principle states that every arriving unit of work demand to a configuration (system) or subconfiguration (subsystem or processor) is matched by a corresponding completion of work there. For finite time intervals this statement only approximates true behavior. Related to this is another conservation principle concerning states of the system, that of *state transition balance*, which we discuss later.

3.2.2 Overlap of Subconfigurations

We assume that no single interaction (job or transaction) overlaps its use of disjoint subconfigurations. In particular, when we consider the subconfiguration to be a processor, a single interaction is present at only one processor at any given instant in time. Note that this depends on some strict definitions of "job" or "transaction" since, for example, all work could be considered as part of the one job called "the operating system." We will provide such definition later under the discussion of the "software unit" entity of software physics.

3.2.3 Single-Step Behavior

This assumption asserts that if we resolve time finely enough we will observe changes in the system only at a single pair of processors, appearing as the movement of a request from one processor to another. Thus we eliminate from consideration the simultaneous "movement" of requests in the system.

3.2.4 Processor Homogeneity

This assumption states that the output rate (power) of a processor is determined only by its queue length and is otherwise unaffected by the arrangement of work elsewhere in the system. This assumption is violated in practice to some extent by the "blocking" of one processor by another. For example as when a disk unit must wait for a buffer to be cleared by a cpu or when occasionally a disk utilizing RPS (rotational position sensing) cannot connect to its channel

for transmission of data, this due to the use of the channel by another disk unit. The homogeneity assumption implies then that the processor is busy if there is work waiting for it. The stochastic counterpart to this assumtption is that interdeparture times at a device are exponentially distributed. In conventional operational analysis, the counterpart is termed device honogeneity, but here we wish to anticipate the possibility of a subconfiguration of equipment being considered a processor.

3.2.5 Routing Homogeneity

This assumption states that the proportion of work arriving to the system that is routed to a given processor may depend only on the multiprogramming level, that is, the concurrent number of interactions in the system. Thus the arrangement of work elsewhere in the system or length of queue at the processor itself does not affect the proportion of work routed there. This assumption, for example, allows for increased work at a paging device with increased multiprogramming level. The stochastic counterpart of this assumption is that job routing follows an ergodic Markov chain.

3.2.6 Decomposition

This principle states that when state transitions between nested subsystems are small in number compared to the number of transitions within the composite, we may reasonably well replace the contained subsystem with an equivalent processor. The characteristics of this equivalent processor are to be determined from a study in isolation of the contained subsystem. We will make use of this principle later in the analysis of an on-line terminal system.

3.2.7 Invariance Assumptions

These are not specific assumptions but rather a type of assumption. When a performance analyst assumes explicitly or otherwise that the change of a given workload or processor parameter will not change any other parameters, the assumption is being made that

those parameters are invariant under the change. Very often it is safe to make such an assumption, but there are exceptions. For example, if through priority setting, jobs of one class, say A, are given preference over jobs of another class, say B, the average multiprogramming level of class B jobs may decrease if the number of class A jobs are increased. Thus it is suggested that these assumptions be made explicit so that unexpected results in model validation can be effectively researched and explained.

SECTION IV

FUNDAMENTAL QUANTITIES

4.0 INTRODUCTION

This section presents the software physics quantities which are the basis for formulating the laws and relationships subsequently given. These quantities are either primary (directly observable) or derived and depend on time, work or the logical interconnection of equipment. They are:

(1) Properties of the *Equipment* and *Implementation* described as the presence of processors with potential performance characteristics.

(2) Properties of the *Logical Equipment Topology*. The description of configurations in software physics; queueing networks.

(3) *Work demand* presented to the configuration by a specific task software unit.

(4) Quantities which describe or predict the *performance* of the system with respect to the tasks defined by work demand.

Our mode of formulation of laws and relationships will generally allow expression in terms of distinct quantities from each of the above groups.

4.1 NOTATION

Software Physics quantities are given in the functional notation:

$$Q(V_1;V_2)$$

where the factor Q is a quality or property and V_1 and V_2 are each lists of designators. The list V_1 designates the software unit, that is, the program and data of the specific execution of a task

or set of tasks. The list V_2 designates the logical subconfigurations in the system. Software units, configurations and logical subconfigurations are more fully discussed in [KOLE76] and [KOVA79].

For the list V_1 (software units) we will use:

Σ	- The full workload.
S or S_i	- Any specific software unit (or member of a collection of software units) as designated by the discussion.

For the list V_2 (equipment) we will use:

ψ	- The full configuration.
χ (or χ_i)	- These designate processors (devices) or collections of them which are characterized by tree structures.
cpu) *disk*) *tape*) *etc.*)	- Devices may be named directly as needed in V_2 descriptions of the configuration.

As the property Q in general describes qualities that depend on the containment of one software unit by another or a piece or class of equipment by another, the lists are made to describe these relationships by given the *contained* unit on the left and the *containing* unit on the right. Thus if $S \supset S_1$, then

$$Q(S_1, S; \chi)$$

is the property Q of the equipment χ operating with software unit S_1 *relative* to the execution of the containing unit S.

As an example, the common notion of the utilization of a device χ is denoted by:

$U(\Sigma;\chi,\psi)$

which is the time of execution of χ with respect to that of ψ on behalf of all software units contained in the full workload.

But

$U(S,\Sigma;\chi,\psi)$

is the utilization (full conditional) of the same equipment with its time of execution counted only when it is on behalf of the software unit S.

Finally, it is convenient to denote the collection of values over all equipment classes of the property q in an array or vector form. Thus

$\underline{q}(\Sigma;\psi)$

denotes the array with elements $q(\Sigma;\chi_1)$, $q(\Sigma;\chi_2)$, etc. where the collection of χ_i constitute the full configuration ψ or some specifically designated set of equipment classes.

A null position indicated in the list V_q by a dot, indicates that the elements of the array are the values of q in *contained* equipment relative to *containing* equipment.

Thus

$\underline{q}(\Sigma;\cdot,\psi)$

is the array of elements $q(\Sigma;\chi_1,\psi)$, $q(\Sigma;\chi_2,\psi)$, etc. with the designations of the χ_i either left arbitrary or defined by the context.

4.2 BASIC PROPERTIES

The quantities of time and work are fundamental to the description of the occurrence of events and performance in an executing system.

4.2.1 Execution Time: $Tx(S;\chi)$

This quantity measures the active time for the software unit designated by S on the subconfiguration (equipment) designated by χ. It may thus be thought as a clock which runs only when the designated software unit and equipment are active. This definition encompasses both the "busy time" and "observational interval" of conventional operational analysis, but differs in that unlike the conventional formulation:

i) Time may be counted on behalf of specific software units as well as the full workload.

ii) $Tx(L;\psi)$, the execution time of the full workload on the full configuration does not count idle time (no constituent active), whereas the conventional "observation interval" may do so.

Note also that the reference to "busy time" above was to that as defined in operational analysis and does not include time that a device is blocked (e.g., a disk in seek or RPS delay). These points are further discussed in [KOVA79].

4.2.2 Work: $W(S;\chi)$

This quantity in software physics describes events accomplished or demanded. Work performed is *operationally* defined to be equal to the number of bytes (character containers of 8 bits) transferred to or from a processor memory. Thus one unit of work is performed for each byte written to a device (or memory). The unit of work has been designated the "WORK", denoted by ω. As with execution time, work is counted on behalf of the software unit and equipment designated by S and χ, respectively.

Thus, for example, $W(S;cpu)$ measures work performed by the cpu (bytes transferred to a memory) on behalf of the software unit S.

Note that performing a unit of software physics work represents an event at a more elementary level than does the "request" of standard queueing theory or operational analysis. The grouping of work into requests or blocks will be considered to be a property of the *implementation*.

4.3 EQUIPMENT AND IMPLEMENTATION PROPERTIES

The quantities in this group describe the *potential* work performance capability of equipment and the implementation parameters on which this capability depends. Processors or collections of them are frequently referred to as configurations, or more generally *subconfigurations*. The topological properties of equipment interconnections is discussed in the next section.

4.3.1 Absolute Power: $P(\cdot;\chi)$, (or $P(\chi)$, $P(S;\chi)$

This is a derived quantity of software physics defined by:

$$P(\cdot;\chi) = \frac{W(\cdot;\chi)}{Tx(\cdot;\chi)}$$

This expression states that any work performed by the device or configuration denoted by χ is to be divided by the time χ is active. Absolute power may be developed using the conditions existing for a specific software unit in which case we define accordingly:

$$P(S;\chi) = \frac{W(S;\chi)}{Tx(S;\chi)}$$

Implementation parameters may be included in the list following χ if desired to indicate that a specific value of say, disk seek time, or record size was used for requests on behalf of S.

Absolute power at the configuration level, i.e.,

$$P(L;\psi) = \frac{W(L;\psi)}{Tx(L;\psi)}$$

can be interpreted as overall system throughput on a byte basis in that it gives the rate of work completion in the system as a whole as measured by the elapsed time clock.

4.3.2 Blocksize Work: $Wb(S;\chi)$

This quantity is an implementation parameter which specifies the quantity of work performed each time a specific interaction visits the subconfiguration χ. It thus corresponds to the work per request when "request" is defined to be an *unbroken* sequence of demands at the device on behalf of one job or transaction user. So it is the amount of work done by the device on behalf of the software unit S, uninterrupted by the execution of another device on behalf of that same software unit.

4.4 PROPERTIES OF THE EQUIPMENT TOPOLOGY

We consider here two conceptions of the way in which the logical interconnection of equipment is described. The first is that of conventional queueing network analysis and describes how the processors form a conceptual *network* as a consequence of the way requests are routed in the system. From this we develop the Buzen Central Server Models (BCSM) for batch and terminal driven systems.

The second conceptualization is that of the logical *configurations* and *subconfigurations* of software physics. This mode of description visualizes the equipment and the paths to it as forming rooted trees. The useful properties derived from this conception include the natural way in which work demand for a task can be partitioned in conformance with the structure of the logical systems. This structuring takes advantage of inclusion properties and so forms the basis for analysis using natural system decompositions.

4.4.1 Queueing Networks and the Routing of Requests

A queueing network is a conceptualization of how the demands of interactions enter the system, circulate from device to device (having received service after possibly entering a queue) and finally complete. This last event is represented by showing that the interaction leaves the system. The transitions from device to device or to exit are associated with probabilities

$q_{i_0}(S)$, where S is the software unit being tracked. We frequently assume that all interactions have the same $q_{i_0}(S)$ and simply write q_{i_0}. The subscript o for i denotes routing to a device from outside the system, while 0 for j denotes the interactions departure from the system. Figure 4.4.1 shows this network conception in general. The dashed line marked "closed" is to model the situation where the number, N, of interactions circulating in the network is held fixed - as soon as one leaves, another takes its place. Closed networks represent systems operating under a backlog.

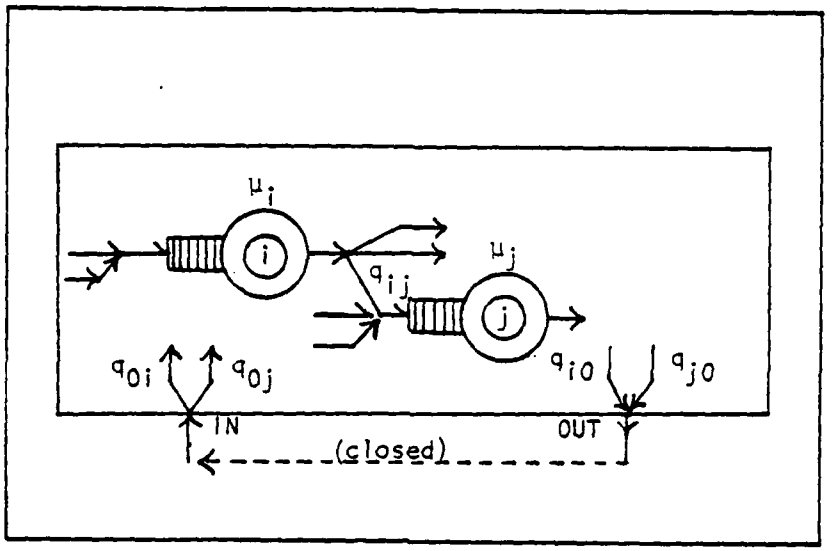


Figure 4.4.1
Devices in a Queueing Network

In open networks, the number of interactions in the network may vary. In these, one generally knows the system throughput and wishes to determine the steady state value of N and the way it distributes to the various devices.

As assumption that is generally made is that the interactions do not overlap their use of devices; that is, work on behalf of an interaction appears in queue or in service at one and only one device at a time. Travel time between devices is assumed

to be zero, but if it is significant as in the case when a long transmission line is in the network, one can substitute a device having a power which gives the appropriate delay time.

Two network configurations of special interest are those based on Buzen's control server model (BCSM). These are so named because one device (the cpu) acts in a pivotal capacity. That is, an interaction always makes a request visit to the cpu before and after making a request of any other device or exiting the system. This is depicted in Figure 4.4.2 which is the batch central server system. In Figure 4.4.3 the central server forms the computer subsystem for a terminal driven system. In this latter configuration, M terminal users are signed on and submit interactions to the central subsystem. When processing is complete, the user spends Z units of "think time" before entering the next interaction. During the processing interval, the user's terminal is blocked, that is, it can enter no other interactions.

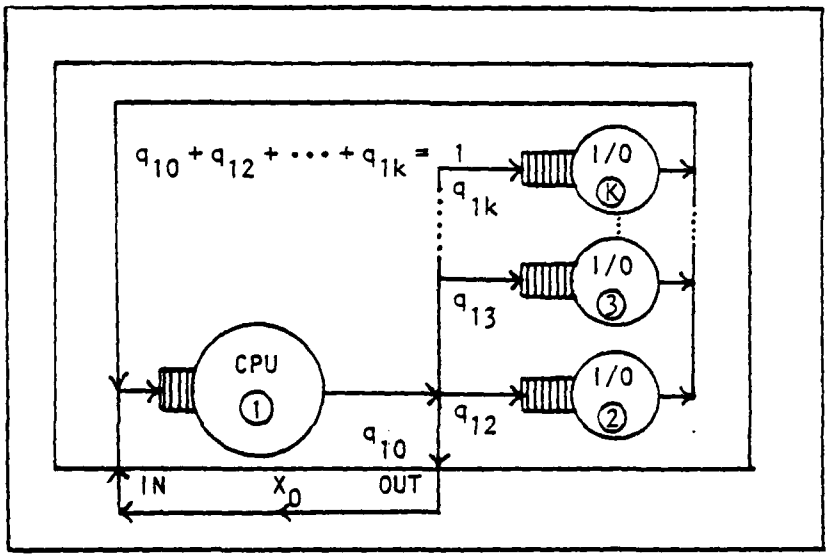


Figure 4.4.2
Buzen's Central Server Network

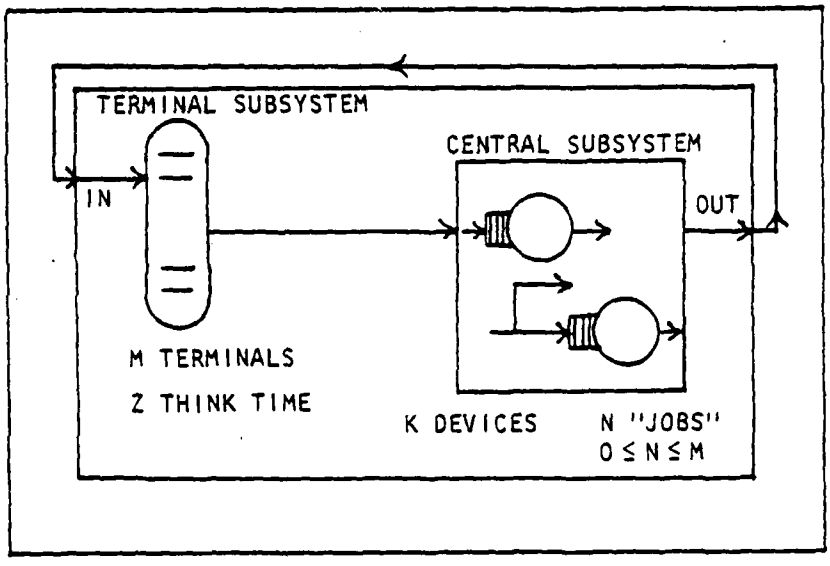


Figure 4.4.3

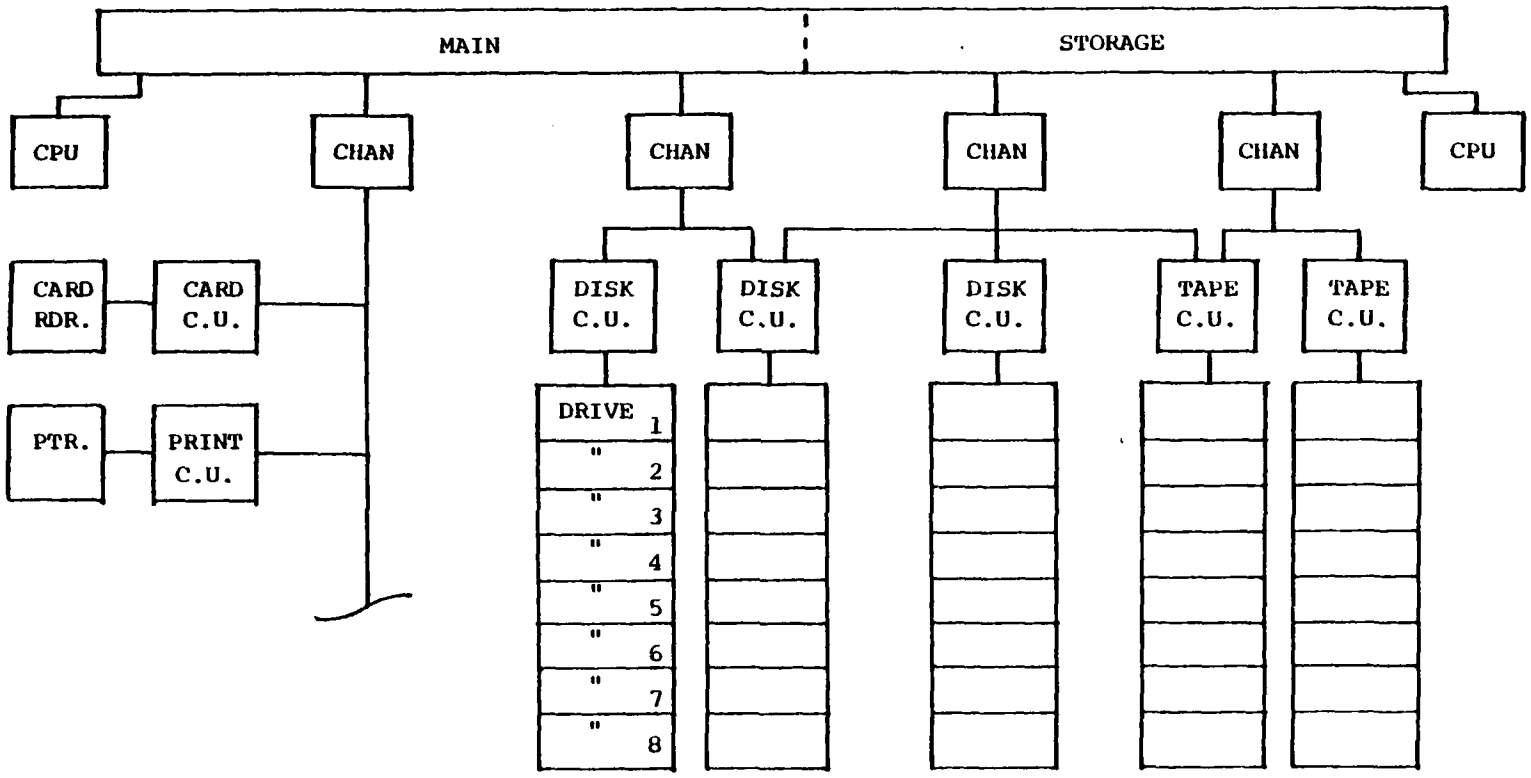
Terminal System with Central Server

4.4.2 Configurations and Subconfigurations in Software Physics

The software physics conception of the logical structure of computing systems is that of rooted trees formed by the graph union of all possible data paths that may occur in the course of some execution. A full discussion of this concept and these properties is given in [KOVA79]; we shall only give a description of a few of these properties here.

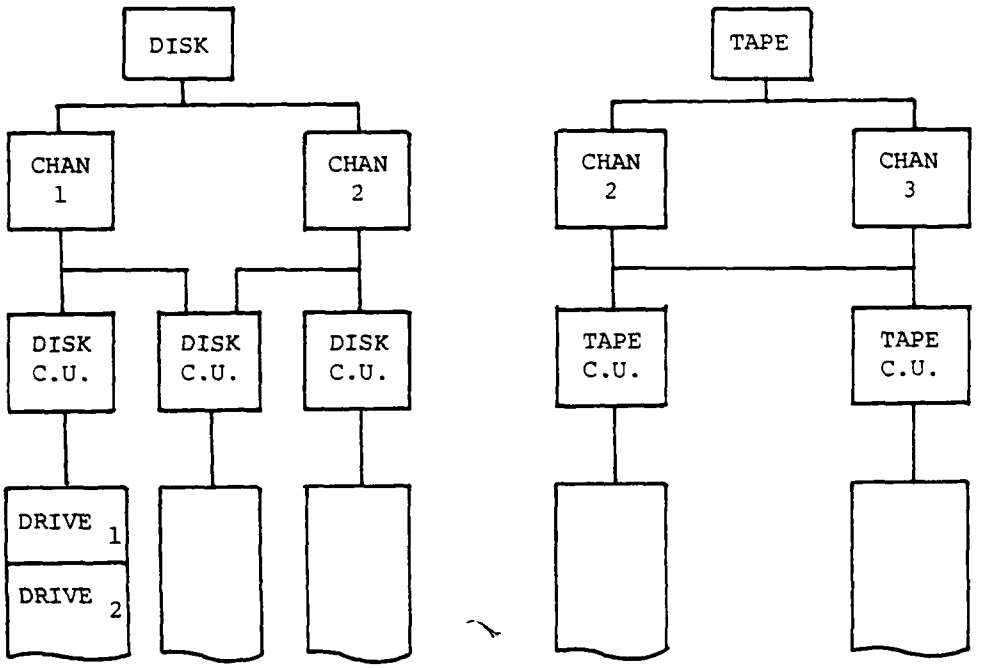
The concept of the configuration is built on the notion of the conventional graph of equipment *interconnection*, an example of which is given in Figure 4.4.4.

Augmentations are next made to develop *logical subconfigurations*. This is done by the operation of graph composition which is the union of instantaneous paths to drives plus the intrusion of a highest level mode, if one does not already exist. For example in order to form the tape logical subconfiguration, one forms the union of all paths to tape drives including the channel and controller devices for them and inserts a node labeled "TAPE." Examples are shown for both disk and tape logical subconfigurations in Figure 4.4.5.



A CONFIGURATION - "STANDARD" REPRESENTATION

Figure 4.4.4



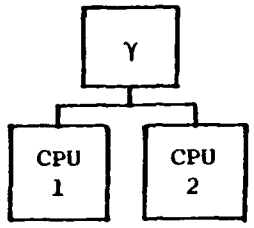
THE DISK SUBCONFIGURATION THE TAPE SUBCONFIGURATION

EQUIPMENT CLASS SUBCONFIGURATIONS

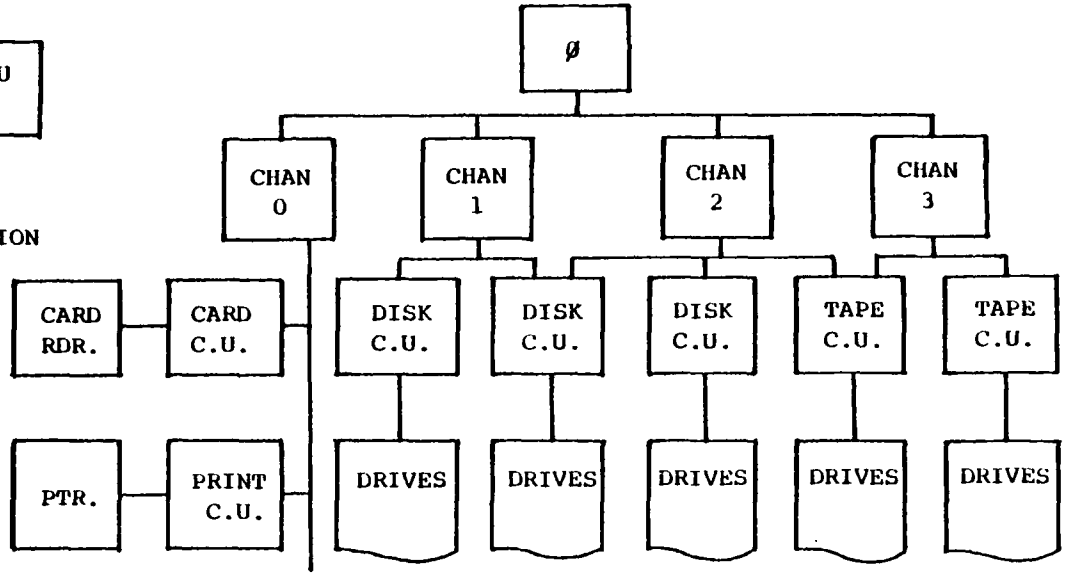
Figure 4.4.5

Most important is that each node defines a relative root which has the upper lattice property, that is, every node contains or covers the properties of all nodes dependent on it. So for example, in Figure 4.4.6 we observe that the "CHAN 2" logical subconfiguration is both a tape and a disk channel subconfiguration. In the same figure, ψ is the entire input/output logical subconfiguration, γ is the cpu logical subconfiguration.

The description of configurations by these rooted tree structures will be useful to us in describing overall how work is distributed to equipment. It is also a natural means for dealing with the decomposition of systems into subsystems and so has application in conventional queueing networks having subsystems as servers.



THE CPU SUBCONFIGURATION



THE INPUT/OUTPUT SUBCONFIGURATION

(EQUIPMENT CLASS = PERIPHERAL DRIVES)

EQUIPMENT CLASS SUBCONFIGURATIONS

Figure 4.4.6

4.4.3 The Configuration and the Bulk Distribution of Work

Recall that the symbol for an arbitrary configuration is χ. When we need to indicate containment of one subconfiguration by another we do this by use of one or more asterisks (*) in the subscript of χ. The asterisk represents a string of one or more subscript values. In a given context, if we use the asterisk to represent a string of subscripts, we must remember that we are referring to that same string whenever an asterisk occurs. Additional asterisks in the subscript thus represent *additional* levels of subscripting (depth in the structure) to that indicated by the first. Thus χ_* refers to a subconfiguration at least one level deeper than χ and χ_{**} is at least one level deeper than χ_* and so on. If we write

$$\sum_j F(S;\chi,\chi_{*j})$$

then we are summing the F's for all subconfigurations that are at least two levels below χ etc. Hierarchies of software units may be handled in the same way.

Software physics describes the demands made on subconfigurations and processors in terms of the quantity of work that each performs. However, we will need to refer to quantities of work demand *arriving* to the system denoted by the vector:

$$\underline{Wa}(S;\psi) = \{Wa(S;\chi_1),\ Wa(S;\chi_2),\cdots\}$$

where the $\{\chi_i\}$ represent a partition of ψ; that is: $\chi_i \cap \chi_j = 0 \quad i \neq j$

$$\psi = \bigcup_j \chi_i$$

The ratios

$$D(S;\chi_i,\psi) = \frac{Wa(S;\chi_i)}{Wa(S;\psi)} \qquad \{\chi_i\} \text{ a partition of } \psi$$

are called the bulk work distributions of the arriving work and are denoted by the vector:

$$\underline{D}(S;\cdot,\psi) = \{D(S;\chi_1,\psi),\ D(S;\chi_2,\psi),\cdots\}$$

Note that we cannot yet say anything about the measured work performed at the χ_i relates to the arriving $Wa(S;\chi_i)$. This must wait for a conservation assumption we will make presently.

A specific quantity of arriving work demand is that related to a single or mean interaction (job, transaction, or the like). This vector quantity is denoted:

$$\underline{Wd}(S;\psi) = \{Wd(S;\chi_1), Wd(S;\chi_2), \cdots\}$$

Again, the magnitude is given by

$$|\underline{Wd}(S;\psi)| = Wd(S;\psi) = \sum_i Wd(S;\chi_i)$$

Now for a given software unit S (an interaction or collection of interactions):

$$\frac{Wd(S;\chi_i)}{Wd(S;\psi)} = \frac{Wa(S;\chi_i)}{Wa(S;\psi)} = D(S;\chi_i,\psi)$$

since the Wa and Wd measure the same things but over different time bases.

In our present discussion we will use the χ_i as if they were devices, remembering that they can just as well be proper subconfigurations. What we leave for a subsequent analysis is how we may <u>in general</u> substitute a subconfiguration for collections of devices (or contained subconfigurations). We will thus be concerned with work distributions of

$$\underline{Wa}(S;\psi) = Wa(S;\psi)\cdot\underline{D}(S;\psi) = \sum_i Wa(S;\chi_i)\cdot\underline{D}(S;\psi)$$

and

$$\underline{Wd}(S;\psi) = Wd(S;\psi)\cdot\underline{D}(S;\psi) = \sum_i Wd(S;\chi_i)\cdot\underline{D}(S;\psi)$$

where the χ_i are *processors* or their equivalents in the system.

The above distributions of work are on a *bulk* basis. That is, they show how work distributes on the average to the devices (or subconfigurations). It will develop later that these will suffice as workload specification in the calculation of the other descriptive quantities we seek.

It is also possible to formulate distribution numbers for proper subconfigurations. This is done thoroughly in [KOVA79], so we only give an indication here. If the quantity of work arriving to a proper subconfiguration configuration χ_* is $Wa(S;\chi_*)$ then,

$D(S;\chi_{**},\chi_*) = \frac{W(S;\chi_{**})}{W(S;\chi_*)}$ is the bulk distribution of work for the subconfiguration χ_* to χ_{**}

4.5 PERFORMANCE QUANTITIES

These quantities tell of the rates at which the given configuration is processing the required workload and consequently of the (response) time that we must wait for an interaction to complete. Intimately connected with these quantities are the *utilizations of devices* which indicate how much of the device capacity we use on behalf of some specific interactions or the total workload and the lengths of queues at the device.

4.5.1 Relative Power: $P(S_i,S;\chi_*,\chi)$ also $P(S;\chi_*,\chi)$ $P(S;L;\psi)$

This quantity is most generally defined as

$$\frac{W(S_i;\chi_*)}{Tx(S;\chi)} \qquad S_i \in S$$

and gives the rate of performing work on behalf of the software unit S_i on the subconfiguration χ_* relative to the clock which counts time when S and χ are active. It is sometimes called throughput power for it gives the relative rate of work unit request completions. This rate must not be confused with the notion of "service rate" from conventional queueing analysis. This latter quantity is the request completion rate relative to the clock which is active only when the server is busy and is therefore more like the software physics quantity of *absolute* power.

The quantity

$$P(S,L;\psi) = \frac{W(S;\psi)}{Tx(L;\psi)}$$

called the software conditional relative power, may be interpreted as the overall work level throughput for the software unit S. It is the work completion rate on behalf of S as measured by the system elapsed time clock.

Notice that

$$P(S,L;\psi) = \frac{\sum_i (S;\chi_i)}{Tx(L;\psi)} = \sum_i P(S,L;\chi_i,\psi)$$

where $\{\chi_i\}$ is a partition of ψ. So the software conditional relative power is the sum of the fully conditional relative powers over a partition of the configuration.

4.5.2 Utilization: $U(S_i,S;\chi_*,\chi)$ also $U(S;\chi_*,\chi)$

This quantity is defined most generally as:

$$\frac{Tx(S_i,\chi_*)}{Tx(S;\chi)} \qquad S_i \in S$$

and gives the ratio of time that the software unit S_i is active on subconfiguration χ_* to the time the containing software unit S is active on a containing subconfiguration. The term "utilization" encountered in conventional parlance is:

$$U(L;\chi_i,\psi) = \frac{Tx(L;\chi_i)}{Tx(L;\psi)}$$

That is, the ratio of the *unconditional* execution time on processor χ_i to the execution time of the full workload on the full configuration.

4.5.3 The System State: Vectors: $\underline{\dot{W}}(S) = [\dot{W}_1, \dot{W}_2, \cdots, \dot{W}_k](S)$

or

$\underline{N}(S) = [n_1, n_2, \cdots n_k](S)$

Scalars: $\dot{W}(S;\psi)$ $N(S;\psi)$

The vector quantity shows how the work or requests in the system for software unit S are distributed at some instant in time to each of the k devices (or subconfigurations). Thus $\dot{W}_i$ units of work are in queue or in service at device i in the work formulation. Similarly n_i requests are in queue or in service at device i in the request formulation. Note that the scalar total waiting work

$$\dot{W}(S;\psi) = \sum_{i=1}^{k} \dot{W}_i(S)$$

or in terms of requests

$$N(S;\psi) = \sum_{i=1}^{k} n_i(S)$$

The work and request forms of these quantities are related by

$$\dot{W}_i(S) = \overline{Wb}(S;\chi_i) \cdot n_i(S)$$

where $\overline{Wb}$ is the average processor blocksize work for the software unit S.

4.5.4 System Throughput: $P(S,L;\psi)$ also $\chi_0(S)$

System Response Time: $Tb(S;\psi)$

Under the conditions of the conservation of work is the configuration the software conditional relative power for the entire configuration

$P(S,L;\psi)$

may be interpreted as throughput on a work level basis for the entire system. This means that for a sufficiently extended period,

$Tx(L;\psi)$ the rate of interaction completion $\chi_0(S)$ (this quantity measured externally as jobs or transactions completed) is given by:

$$\chi_0(S) = \frac{W(S;\psi)}{Wd(S;\psi)} \cdot \frac{1}{Tx(L;\psi)}$$

$$= \frac{P(S,L;\psi)}{Wd(S;\psi)}$$

System response time is that busy time measured by the clock at level ψ for an interaction to enter the system, make visits to the servers (and wait in queues, if necessary) and finally exit the ψ configuration. It is the time that the ψ configuration is busy with (including delay <u>in</u> ψ) an $S_i \in S$ interaction. It is identical to the response time, R, of conventional operational analysis.

SECTION V

LAWS AND RELATIONS

5.0 INTRODUCTION

We now present some relations between the defined quantities of the last section. All the demonstrations are operational in the sense that no stochastic assumptions are made and that the quantities are to be obtained from direct measurement over finite observation intervals. Included are immediate relations between the fundamental quantities as well as work flow relations derived from consideration of entire systems in balanced flow.

5.1 General Laws and Relations

These are valid in both balanced and non-balanced states of a system, that is, they depend only on work having arrived to a server and not on any conservation principles over the network as a whole.

5.1.1 Utilization Law

From the definitions for utilization

$$U(S,L;\chi_i,\psi) = \frac{Tx(S;\chi_i)}{Tx(L;\psi)}$$

and for relative and absolute powers, we have:

$$P(S,L;\chi_i,\psi) = \frac{W(S;\chi_i)}{Tx(L;\psi)} = \frac{W(S;\chi_i)}{Tx(S;\chi_i)} \cdot \frac{Tx(S;\chi_i)}{Tx(S;\psi)}$$

from which

$$P(S,L;\chi_i,\psi) = P(S;\chi_i) \cdot U(S,L;\chi_i,\psi)$$

and so

$$U(S,L;\chi_i,\psi) = \frac{P(S,L;\chi_i\psi)}{P(S;\chi_i)} \quad \text{(utilization law)}$$

when $S = L$, we have more simply:

$$U(L;\chi_i,\psi) = \frac{P(L;\chi_i,\psi)}{P(L;\chi_i)}$$

5.1.2 Little's Law

Enormously useful in the analysis of queueing behavior is Little's Law, so named for it was proved under very general conditions by J.D.C. Little in 1961. However, it existed for many years before that as what Kleinrock calls a "folk theorem." It relates response time of a system (or device or queue itself) to the amount of work waiting. The relationship is:

$$T_w(S;\chi_i) = \frac{\overline{w}_w(S;\chi_i)}{P(S,L;\chi_i,\psi)} \qquad (5.1.1)$$

where $T_w(S;\chi_i)$ is the mean waiting time at subconfiguration χ_i and $\overline{w}_w(S;\chi_i)$ is the mean amount of work waiting for or in service at χ_i.

The conventional operational analysis formulation is:

$$R_i = \frac{\overline{n}_i}{\chi_i}$$

where $\overline{n}_i$ is the mean number of *requests* waiting for or in service at device i and χ_i is the request completion rate.

To show the software physics formulation of this result, consider that we observe and count the work waiting for or in service (called the waiting work) as a function of time.

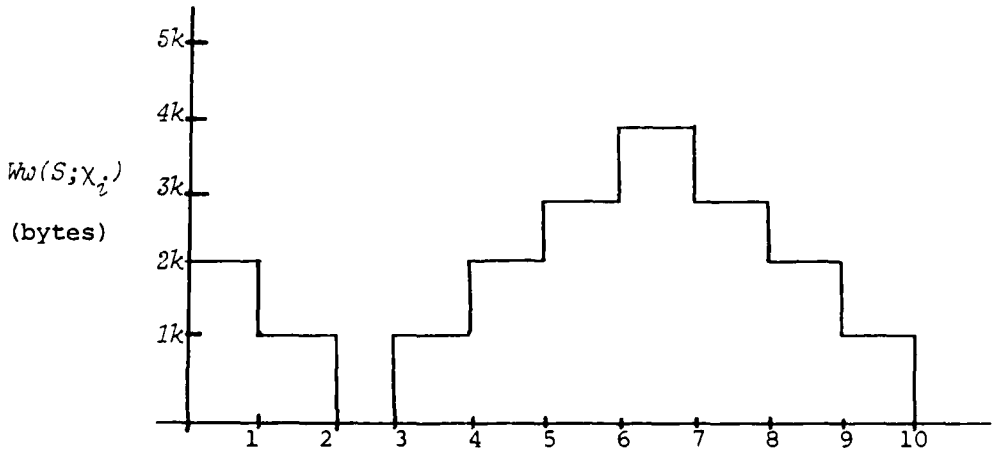


Figure 5.1.1

Waiting Work at A Subconfiguration
(Example)

Now if we let $A(S;\chi_i)$ be the area under the graph of waiting work, we write:

$$A(S;\chi_i) = \sum_{t=0}^{Tx(L;\psi)} Ww(S;\chi_i)\Delta t$$

The average height of the graph is:

$$\overline{Ww}(S;\chi_i) = \frac{A(S;\chi_i)}{Tx(L;\psi)}$$

This is the average amount of work in the system. Note that we use the elapsed time $Tx(L;\psi)$ as our clock.

Now the average completion rate is $\delta(S;\chi_i)$, the work done divided by the elapsed time or

$$\delta(S;\chi_i) = \frac{W(S;\chi_i)}{Tx(L;\psi)} = P(S,L;\chi_i,\psi)$$

So the average time that a unit of work demand spends in the system is:

$$Tw(S;\chi_i) = \frac{\overline{Ww}(S;\chi_i)}{\delta(S;\chi_i)} = \frac{\overline{Ww}(S;\chi_i)}{P(S,L;\chi_i,\psi)}$$

Notice that $Tw(S;\chi_i)$ is not a rate per unit of work but a *time*. The relationship tells us nothing about *how* the work arrives to or departs from the system. In particular, if work arrives in batched *requests* and is served under a first-come-first-served (FCFS) discipline, all unit work demands in the request wait for their service *concurrently*.

As an example, consider the graph of Figure 5.1.1. The accumulated waiting time over the 10 second period is 19Kw seconds. So the average waiting work $\overline{Ww}(S;\chi_i)$ is 1.9 Kw. The relative power is the work completed divided by the same 10 seconds, i.e.,

$$P(S,L;\chi_i,\psi) = \frac{W(S;\chi_i)}{Tx(L;\psi)} = \frac{6 \times 10^3}{10} = 600 \text{ W/sec.}$$

(there were 6 - 1KB completions)

Consequently the mean waiting time at χ_i is

$$Tw(S;\chi_i) = \frac{1.9 \text{ Kw}}{600 \text{ w/s}} \text{ or about 3.17 seconds.}$$

5.2 FLOW BALANCE AND THROUGHPUTS

We now discuss a work conservation principle at the configuration and system state transition levels. This principle is called one of flow balance for as a consequence processing rates of individual subconfigurations are related to each other and to the interaction level rate of processing (called the throughput).

5.2.1 Configuration Work Flow Balance

At a macroscopic level we observe the arrival of a vector of work demand for a set of interactions $\underline{Wa}(S;\psi)$ and now wish to establish a relationship between this quantity and the work performed inside the system. What we require is that for large enough values of $Tx(L;\psi)$ (elapsed time), the performed work of a subconfiguration should be very nearly the same as the total observed to arrive times the bulk distribution quantity for that subconfiguration. That is:

$$|W(S;\chi_i) - Wa(S;\psi) \cdot D(S;\chi_i,\psi)| << W(S;\chi_i)$$

for any subconfiguration (or device) χ_i where

$$Wa(S;\psi) = \sum_{*} Wa(S;\chi_*) \text{ for any partition } \{\chi_*\} \text{ of } \psi$$

So, approximately:

$$|W(S;\chi_i) - Wa(S;\psi) \cdot D(S;\chi_i,\psi)| = 0$$

This conservation assumption leads us to the principle of *configuration work flow balance* :

$$\frac{Wa(S;\psi)}{Tx(L;\psi)} = \frac{W(S;\chi_i)}{Tx(L;\psi)} \frac{1}{D(S;\chi_i,\psi)}$$

or

$$Pa(S,L;\psi) = \frac{P(S,L;\chi_i\psi)}{D(S;\chi_i,\psi)} \quad \text{for any subconfiguration } \chi_i \tag{5.2.1a}$$

Since

$$\sum_i D(S;\chi_i,\psi) = 1$$

and

$$\sum_i P(S,L;\chi_i,\psi) = P(S,L;\psi)$$

we may rewrite the right hand side of (5.2.1a) without a subscript giving:

$$P(S,L;\psi) = \frac{P(S,L;\chi_i,\psi)}{D(S;\chi_i,\psi)} \tag{5.2.1b}$$

for any subconfiguration χ_i.

We may now interpret $P(S,L;\psi)$ as the work level throughput of the configuration and the $P(S,L;\chi_i\psi)$ as the work level throughputs of the χ_i. To develop a throughput at the interaction level we recall that each interaction requires work:

$$Wd(S;\psi) = \sum_i Wd(S;\chi_i)$$

Multiplying (5.2.1b) on both sides by this amount of work we obtain:

$$P(S,L;\psi)\cdot Wd(S;\psi) = \frac{P(S,L;\chi_i,\psi)}{Wd(S;\chi_i)} \tag{5.2.2a}$$

or

$$X_0(S) = \frac{P(S,L;\chi_i,\psi)}{Wd(S;\chi_i)} \tag{5.2.2b}$$

for any subconfiguration χ_i where $\chi_0(S)$ is the interaction level throughput for interactions belonging to S and χ_i is any subconfiguration (or device) belonging to ψ.

Equations (5.2.2a,b) tell us what device level throughput must be achieved at each device in order to sustain a configuration work throughput, $P(S,L;\psi)$, or interaction throughput, $\chi_0(S)$, and so are called the Forced Flow Laws.

5.2.2 Flow Balance in General Queueing Networks

In traditional formulations of queueing networks, work demands for a given job are imagined to circulate from device to device with routing determined by the q_{ij} and job entry or exit is thought of as from or to device zero. Conservation of *transition* equations are written at each node to express flow balance. That is, for a request size $Wb(S;\chi_i)$

$$\frac{W(S;\chi_j)}{Wb(S;\chi_j)} = \sum_{i=0}^{k} \frac{W(S;\chi_i)}{Wb(S;\chi_i)}\, q_{ij} \qquad \begin{matrix}(5.2.3a)\\ j = 0,\cdots,k\end{matrix}$$

This is an expression of the fact that the number of requests completed at χ_j is the same as the total number which arrived from all sources connected (in the operational sense) to j. From the definition of relative power, we get after dividing both sides of (5.2.3a) by $Tx(L;\psi)$:

$$\frac{P(S,L;\chi_j,\psi)}{Wb(S;\chi_j)} = \sum_{i=0}^{k} \frac{P(S,L;\chi_i,\psi)}{Wb(S;\chi_i)}\, q_{ij} \qquad \begin{matrix}(5.2.3b)\\ j = 0,\cdots,k\end{matrix}$$

whence:

$$P(S,L;\chi_j,\psi) = \sum_{i=0}^{k} P(S,L;\chi_i,\psi)\cdot\frac{Wb(S;\chi_j)}{Wb(S;\chi_i)}\, q_{ij} \qquad \begin{matrix}(5.2.3c)\\ j = 0,\cdots,k\end{matrix}$$

We note that the quantity $P(S,L;\chi_0,\psi)$ and $Wb(S;\chi_0)$ represent rates and events at the external *interaction* level and are given by:

$$P(S,L;\chi_0,\psi) = P(S,L;\psi) = \sum_{i=1}^{k} P(S,L;\chi_i,\psi)$$

$$Wb(S;\chi_0) = Wd(S;\psi) = \sum_{i=1}^{k} Wd(S;\chi_i)$$

The expressions (5.2.3c) are called the Flow Balance Node Equations. The quantity

$$\frac{P(S,L;\psi)}{Wd(S;\psi)} = \frac{\sum_{i=1}^{k} P(S,L;\chi_i,\psi)}{\sum_{i=1}^{k} Wd(S;\chi_i)}$$

is just the external interaction throughput $\chi_0(S)$ when $Tx(L;\psi)$ is sufficiently large.

In Buzen's operational analysis [DENN78], the ratios of device to external interaction throughput are defined as *visit ratios* by

$V_i = X_i/X_0$ where the X_i are the device request completion rates.

Here the analogous quantities are obtained by taking the ratios of device relative power with the software conditional ψ-level power, that is:

$$V(S;\chi_i) = \frac{P(S,L;\chi_i,\psi)}{P(S,L;\psi)} = \frac{W(S;\chi_i)}{W(S;\psi)} = D(S;\chi_i,\psi) \qquad (5.2.4a)$$

Equation (5.2.4a) shows that each byte of work completed by requires $D(S;\chi_i,\psi)$ bytes completed by χ, that is, V is a "per byte" visit ratio. For a completion at the *interaction* level we observe a visit ratio:

$$V^*(S;\chi_i) = V(S;\chi_i)\cdot Wd(S;\psi) = D(S;\chi_i,\psi)\cdot Wd(S;\psi)$$
$$= Wd(S;\chi_i) \qquad (5.2.4b)$$

That is, the quantity $Wd(S;\chi_i)$, the work demand, is the same quantity as the number of visits for the transaction or job event.

Now, Equation 5.2.3b can be divided by $P(S,L;\psi)$ on each side to give:

$$D(S;\chi_j,\psi) = \frac{Wb(S;\chi_j)}{Wd(S;\psi)} q_{0j} + \sum_{i=1}^{k} D(S;\chi_i,\psi) \frac{Wb(S;\chi_j)}{Wb(S;\chi_i)} q_{ij} \qquad (5.2.5)$$

$$j = 0,\cdots,k$$

Equations (5.2.5) are called the Work Distribution Node Equations and are analogous to the Visit Ratio Equations of conventional Operational Analysis. It is important to note, however, that the right hand side of (5.2.5) keeps workload quantities (the distribution numbers) and implementation quantities (the blocksize work and routing frequencies) separate, thereby suggesting analyses where each may be varied independently. The independence of these quantities is valid only approximately as, in reality, the distribution numbers may be altered to some extent by changes in the blocksize work (such as when cpu overheads are reduced by increases in blocksizes).

As a practical matter, we should observe that even if the distribution numbers and blocksize work quantities are known, the $(K+1)^2$ routing frequencies cannot be found in the general network since the equations (5.2.5) are only $K+1$ in number. Fortunately we will not be required to do so, for our invocation of the product from solution later on will require us to provide only the interaction work demand (or distribution numbers) and the absolute subconfiguration powers.

The main use of the Work Distribution Node Equations is in fact to prove the validity of the product form solution. So for the distribution numbers themselves, and as a consequence to interaction work demand values, they may be derived from an analysis of the workload or approximated by measurement of the workload in execution.

5.2.3 A Special Case - The Buzen Central Server Model (BCSM)

It turns out that for the BCSM (see Figure 4.4.2), the Flow Balance and Work Distribution Node Equations (5.2.3c and 5.2.5) simplify greatly and can be solved for the q_{ij}. Note first that:

$$q_{01} = 1 \qquad q_{0i} = 0 \qquad i \neq 1$$

$$q_{1j} = 1$$

$$q_{i0} = 0 \qquad i \neq 1$$

Now the Flow Balance Equations are:

$$P(S,L;\psi) = P(S,L;\chi_0,\psi) = P(S,L;\chi_i,\psi)\,\frac{Wb(S;\chi_0)}{Wb(S;\chi_1)}\,q_{10} \tag{5.2.6a}$$

$$P(S,L;\chi_1,\psi) = \sum_{i=0}^{k} P(S,L;\chi_i,\psi)\,\frac{Wb(S;\chi_1)}{Wb(S;\chi_i)} \tag{5.2.6b}$$

$$P(S,L;\chi_i,\psi) = P(S,L;\chi_1,\psi)\,\frac{Wb(S;\chi_i)}{Wb(S;\chi_1)}\,q_{ij} \tag{5.2.6c}$$

$$i = 2,\cdots,k$$

Substituting

$$P(S,L;\psi) = P(S,L;\chi_0,\psi) = \frac{P(S,L;\chi_i,\psi)}{D(S;\chi_i,\psi)} \qquad \text{we get:}$$

$$1 = \frac{Wb(S;\chi_0)}{Wb(S;\chi_1)}\,q_{10} = \frac{Wd(S;\psi)}{Wb(S;\chi_1}\,q_{10} \tag{5.2.7a}$$

$$D(S;\chi_1,\psi) = \frac{Wb(S;\chi_1)}{Wd(S;\psi)} + \sum_{i=2}^{k} D(S;\chi_i,\psi)\,\frac{Wb(S;\chi_1)}{Wb(S;\chi_i)} \tag{5.2.7b}$$

$$D(S,\chi_i,\psi) = D(S;\chi_1,\psi)\,\frac{Wb(S;\chi_i)}{Wb(S;\chi_1)}\,q_{1i} \qquad i = 2,\cdots,k \tag{5.2.7c}$$

And from these we get the routing frequencies:

$$q_{10} = \frac{Wb(S;\chi_i)}{Wd(S;\psi)} \quad \text{where} \quad Wd(S;\psi) = \sum_{*} Wd(S;\chi_*)$$

$\{\chi_*\}$ a partition of ψ

$$q_{1i} = \frac{D(S;\chi_i,\psi)}{D(S;\chi_1,\psi)}\cdot\frac{Wb(S;\chi_1)}{Wb(S;\chi_i)} \qquad i = 2,\cdots,k$$

Notice that substituting this last expression for q_{ij} into (5.2.6c) gives the same result obtained from the Configuration Work Flow Balance Principle, namely:

$$\frac{P(S,L;\chi_1,\psi)}{D(S;\chi_1,\psi)} = P(S,L;\psi) = \frac{P(S,L;\chi_j,\psi)}{D(S;\chi_j,\psi)} \qquad i = 2,\cdots,k$$

Although these Configuration Work Flow Balance properties hold in any node flow balanced network, they are sufficient for the derivation of the Node Flow Balance Equations only in networks which preserve the rooted tree type structures of software physics. The BCSM does in fact do this, if one allows the cpu configuration to include the input/output devices as a subconfiguration. We can therefore directly write the node balance equations for the BCSM:

$$P(S,L;\chi_i,\psi) = \frac{D(S;\chi_i,\psi)}{D(S;\chi_1,\psi)} \; P(S,L;\chi_1,\psi) \qquad i = 2,\cdots,k \tag{5.2.8}$$

5.3 RESPONSE TIME LAWS

5.3.1 General Response Time

If we apply Little's Law to a configuration as a whole, we may write the response time for a *single byte* of work demand in an interaction when there are $\overline{N}$ interactions active for software unit S:

$$Tb(S_i;\psi) = \frac{\overline{N}}{P(S,L;\psi)} \qquad S_i \in S \tag{5.3.1}$$

Since

$$\overline{N} = \sum_j \overline{n}_j \; \overline{n}_1(S)$$

and multiplying numerator and denominator of (5.3.1) by distribution numbers $D(S;\chi_j,\psi)$:

$$Tb(S_i;\psi) = \sum_j \frac{\overline{n}_j}{P(S,L;\chi_j,\psi)} \cdot D(S;\chi_j,\psi)$$

$$= \sum_j t[(S;\chi_j)(\overline{n}_j)] \cdot D(S;\chi_j,\psi) \tag{5.3.2}$$

with $t[\;]$ being the per byte queue plus service time at the indicated subconfiguration

What this says in words is that the formulated scheduling here is such that each byte of interaction work demand for each server experiences a delay plus service time equal to that when there are $\bar{n}_{\chi_j}$ bytes (*for software unit* S) in the queue.

For an entire interaction with work demand $Wd(S;\psi)$:

$$Tb(S_i;\psi) = \sum_j t[(S;\chi_j)(\bar{n}_j)]\cdot D(S;\chi_j,\psi)Wd(S;\psi)$$

$$= \sum_j t[(S;\chi_j)(\bar{n}_j)]\cdot Wd(S;\chi_j) \qquad (5.3.3)$$

Again, we may interpret (5.3.3) as giving the total response time in terms of a "homogenized" product of a response time per byte (dependent on the multiprogramming level at each server) and byte level visits per interaction. This formulation may be interpreted as for a round robin (RR) scheduling algorithm approximating processor sharing.

Returning to Equation (5.3.2) and multiplying by the work per interaction, $Wd(S;\psi)$, and by unity in the form of $Wb(S;\chi_j)/Wb(S;\chi_j)$:

$$Tb(S_i;\psi) = \sum_j \frac{\bar{n}_j}{P(S,\Sigma;\chi_j,\psi)}\cdot D(S;\chi_j,\psi)\cdot Wd(S;\chi_j)\cdot\frac{Wb(S;\chi_j)}{Wb(S;\chi_j)}$$

$$= \sum_j \frac{\bar{n}_j\cdot Wb(S;\chi_j)}{P(S,\Sigma;\chi_j,\psi)}\cdot\frac{Wd(S;\chi_j)}{Wb(S;\chi_j)} \qquad (5.3.4a)$$

$$= \sum_j Tw(S;\chi_j)\cdot\frac{Wd(S;\chi_j)}{Wb(S;\chi_j)} \qquad (5.3.4b)$$

For the right hand side of Equations (5.3.4b), the first factor is the request waiting time at χ_j when there are $\bar{n}_j$ *requests* there and the second term is the number of times requests for an $S_i \in S$ appear at χ_j, that is, the visit ratio of conventional operational analysis. Equations (5.3.4) give precisely the same waiting time

(interaction response time) as does (5.3.3), differing only in the multiplication by unity and the rearrangement of terms. This form of the equation for $Tb(S;\psi)$ can be interpreted as for a first-come-first-served (FCFS) scheduling algorithm at the *request* level, each request being of size $Wb(S;\chi_i)$.

5.3.2 Interactive Response Time

By application of Little's Law to an interactive system we can derive an expression for the response time for an interaction (here a *transaction*). Referring to Figure 5.3.1, we note that the

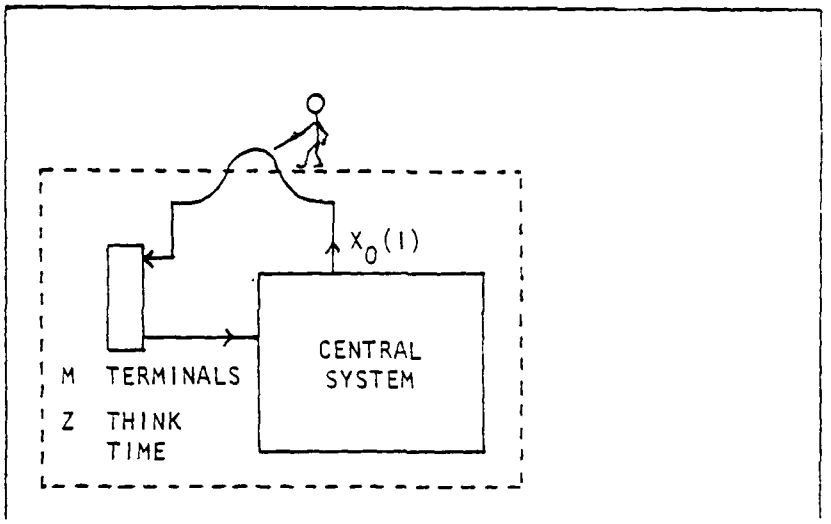


Figure 5.3.1

Observer at an Interactive System

outside observer "sees" the time for an interaction to make a complete circuit as $Tb(S_i;\psi) + Z$, where Z is the "think" time of each signed-on user. That is, if the terminals and the central system are configuration ψ', then the observer measures:

$$Tb(S_i;\psi') = Tb(S_i;\psi) + Z$$

If the number of concurrent users (terminals signed on) is N then we have by Little's Law:

$$Tb(I;\psi') = \frac{M}{P(I,L;\psi')} \cdot Wd(I;\psi') \qquad (5.3.5a)$$

$$= \frac{M}{X_0(I)} \qquad (5.3.5b)$$

Or as seen by the terminal subsystem:

$$Tb(I;\psi) = \frac{M}{X_0(I)} - Z \qquad (5.3.6a)$$

$$= M \cdot \frac{Wd(I;\chi_i)}{P(I,L;\chi_i,\psi)} - Z \quad \text{any} \quad \chi_i \in \psi \qquad (5.3.6b)$$

Equations (5.3.6) are called the Interactive Response Time Formulas.

Note that since we must have:

$$Tb(I;\psi) \geq 1/X_0(I):$$

it follows that:

$$M - ZX_0(I) \geq 1 \qquad (5.3.7)$$

SECTION VI

MULTIPROGRAMMING LOAD ANALYSIS

6.0 INTRODUCTION

In this section we will demonstrate queueing network analysis for configurations which are of the type having product form solutions. These separable queueing networks were originally studied as stochastic exponential server entities by Jackson, Gordon and Newell and others. Buzen and Denning [BUZE77] have demonstrated similar properties for networks with "operational" assumptions. Reiser [REIS79] points out that queueing networks with product form solutions are robust with regard to routing and service time distributions, that is, it is the mean values that dominate the solution.

We will investigate both approximate and exact methods for closed systems. The approximate method is based on a consideration of asymptotic behavior and has been called "bottleneck analysis" by Buzen, a particularly appropriate term, for it is the behavior of the bottleneck device in a separable queueing network which acts as the limiting resource. The method is most useful for quickly determining the effectiveness of changes to the equipment or work demands under light or heavy levels of multiprogramming. The exact methods implemented through efficient algorithms provide throughput functions for queueing networks at any level of multiprogramming and as a consequence provide the basis for an appealing technique for the analysis of terminal driven systems.

6.1 BOTTLENECK ANALYSIS

In this section we show how throughput in closed systems varies with increasing multiprogramming load under the assumptions that work demand and subconfiguration powers remain invariant.

6.1.1 The Bottleneck Subconfiguration

Since

$$U(S,L;\chi_i,\psi) = \frac{P(S,L;\chi_i,\psi)}{P(S;\chi_i)} = \frac{W(S;\chi_i)}{\frac{Tx(S;\psi)}{P(S;\chi_i)}}$$

we have the ratio of utilizations for χ_i, χ_j:

$$\frac{U(S,L;\chi_i,\psi)}{U(S,L;\chi_j,\psi)} = \frac{W(S;\chi_i)}{Tx(S;\psi)P(S;\chi_i)} \cdot \frac{Tx(S;\psi)P(S;\chi_j)}{W(S;\chi_j)}$$

$$= \frac{W(S;\chi_i)}{P(S;\chi_i)} \cdot \frac{P(S;\chi_j)}{W(S;\chi_j)} \qquad (6.1.1)$$

Since the ratio of utilizations is expressed in terms of load invariant quantities (by assumption), the ratio itself is invariant with multiprogramming load. Also, the subconfiguration (say χ_i) with the largest value of

$$\frac{W(S;\chi_i)}{P(S;\chi_i)} \quad \text{, over some time period } Tx(S;\chi_i)$$

has the highest utilization under <u>any</u> system load. It will thus be the first subconfiguration to attain 100% utilization when the load is sufficiently heavy. From the forced flow law applied to this device, we observe:

$$X_0(S) = \frac{P(S,L;\chi_i,\psi)}{Wd(S;\chi_i)}$$

$$= \frac{P(S;\chi_i)}{Wd(S;\chi_i)} \cdot U(S,L;\chi_i,\psi) \rightarrow \frac{P(S;\chi_i)}{Wd(S;\chi_i)} \qquad (6.1.2)$$

$$\text{as} \quad U(S,L;\chi_i,\psi) \rightarrow 1$$

Equation (6.1.2) shows that <u>system</u> throughput is limited by the value

$$\frac{P(S;\chi_i)}{Wd(S;\chi_i)}$$ as the χ_i subconfiguration *saturates*, that is, approaches 100% utilization. This limiting action motivates

calling such a subconfiguration a *bottleneck*; note that there may be more than one in a configuration. For a given collection of subconfigurations $\{\chi_i\}$ which are a partition of ψ denote the bottleneck by a subscript "b" and now $\chi_b(S)$ is the subconfiguration for which

$$\frac{Wd(S;\chi_i)}{P(S;\chi_i)} \text{ is a maximum.}$$

Thus $$\frac{Wd(S;\chi_b)}{P(S;\chi_b)} = MAX\ \{\frac{Wd(S;\chi_b)}{P(S;\chi_b)}\}$$

Note that this choice is software unit dependent, for in a configuration supporting multiple software units it is possible that

$$\chi_b(S) \neq \chi_b(T) \text{ for } S \neq T$$

Now as a <u>single</u> interaction makes its way through the configuration, we have that the total time spent at any subconfiguration χ_i (denoted here by $Txd(S;\chi_i)$ is:

$$Txd(S;\chi_i) = \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

This is valid regardless of how many separate visits are made to each χ_i on behalf of any single interaction in S (denoted S_1).

Now for all the subconfigurations χ_i, the response time is:

$$Tb(S_1;\psi)min = \sum_{i=1}^{k} \frac{Wd(S;\chi_i)}{P(S;\chi_i)} = \sum_{i=1}^{k} Txd(S_1;\chi_i)$$

and the corresponding throughput derived using Little's Law for $\overline{N} = 1$ is:

$$X_0(S) = \frac{1}{Tb(S_1;\psi)} = 1 \Big/ \sum_{i=1}^{k} \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

We can now sketch a curve for throughput as in Figure 6.1.1

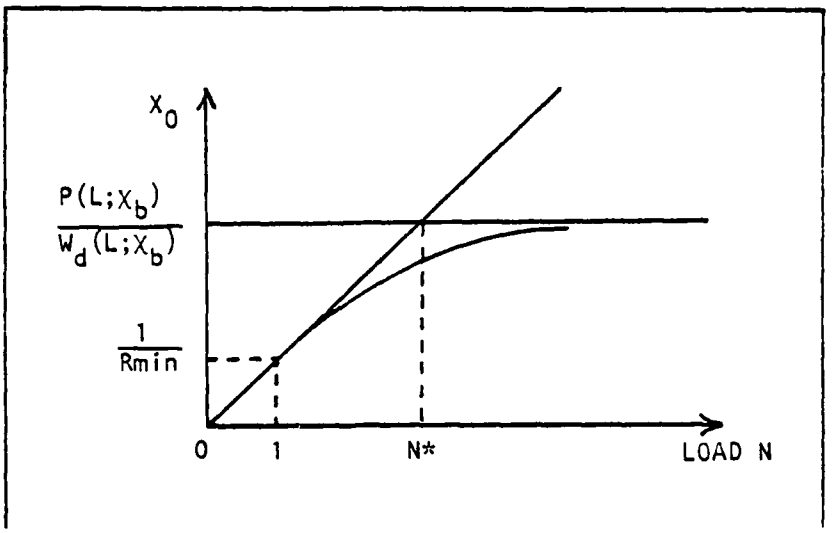


Figure 6.1.1
Bottleneck Analysis - Throughput

One asymptote is the horizontal line for the bottleneck's limiting throughput of

$$\frac{P(S;X_b)}{Wd(S;X_b)}.$$

The other asymptote is the line commencing at the origin and passing through $1/Tb(S;\psi)$ when N, the multiprogramming load is 1.

The approximate throughput function commences at load $N = 1$ with corresponding throughput $1/Tb(S_1;\psi)_{min}$ rising monotonically and staying below the asymptote with slope $1/Tb(S_1;\psi)_{min}$ and approaching (with increasing N) the horizontal asymptote drawn through throughput value

$$\frac{P(S;X_b)}{Wd(S;X_b)}.$$

The load value N^* where the two asymptotes intersect is where

$$\frac{N^*}{Tb(S_1;\psi)_{min}} = \frac{P(S;X_b)}{Wd(S;X_b)}$$

i.e.,

$$N^* = \frac{P(S;\chi_b)}{Wd(S;\chi_b)} \cdot Tb(S_1;\psi)_{min} = \frac{P(S;\chi_b)}{Wd(S;\chi_b)} \cdot \sum_{i=1}^{k} \frac{Wd(S;\chi_i)}{P(S;\chi_i)} \qquad (6.1.2)$$

This load is called the system saturation point [KLEI76] and is given significance in that $N > N^*$ implies that interactions in the system are causing mutual delays through queueing. That is,

N^* is exactly the maximum number of perfectly scheduled interactions that would cause no mutual interference. This is because the fraction of time required at the bottleneck device for a single interaction compared to its total service time is

$$\frac{Txd(S;\chi_b)}{Tb(S_1;\psi)_{min}}$$

where

$$Txd(S;\chi_b) = \frac{Wd(S;\chi_b)}{P(S;\chi_b)}$$

This implies that

$\frac{Tb(S_1;\psi)_{min}}{Txd(S;\chi_b)}$ similar interactions could be scheduled at χ_b without causing interference with each other, and this is exactly what is given by Equation (6.1.2)

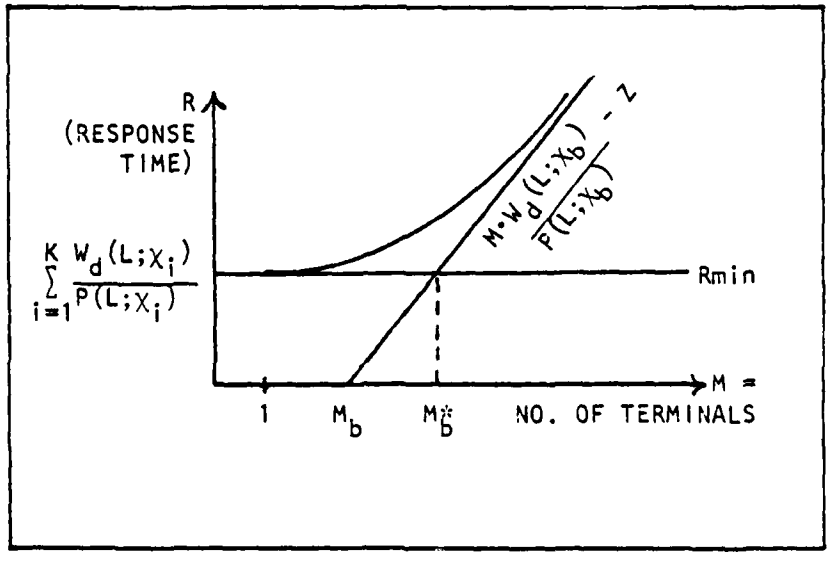


Figure 6.1.2

Bottleneck Analysis - Response Time

As one dare not assume that such perfect scheduling at subconfigurations is in fact realized even for $N < N^*$, the curve sketched remains below the sloping asymptote and the throughputs $N/Tb(S_1;\psi)_{min}$ are not achieved.

A similar type of analysis, but in terms of response time, for terminal driven systems leads to a sketch as in Figure 6.1.2.

As before, the minimum 1-interaction response time is given by:

$$Tb(S_1;\psi)_{min} = \sum_{i=1}^{K} \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

and this is directly plotted as the horizontal asymptote. Next we recall that when saturated, the bottleneck subconfiguration χ_b limits throughput to

$\frac{P(S;\chi_b)}{Wd(S;\chi_b)}$ when χ_b is saturated.

From this and the Interactive Response Time Law we have:

$$Tb(S_1;\psi) = \frac{M}{X_0(S)} - Z \geq M \cdot \frac{Wd(S;\chi_b)}{P(S;\chi_b)} - Z = Tb(S_1;\psi)_a \qquad (6.1.3)$$

and the right-hand side of the inequality is the equation for the slanting asymptote in Figure 6.1.2.

The intersection of this asymptote with the M-axis is at:

$$Mb = Z \cdot \frac{P(S;\chi_b)}{Wd(S;\chi_b)} = \frac{Z}{Txd(S;\chi_b)} \qquad (6.1.4a)$$

The intersection with the $Tb(S_1;\psi)_{min}$ asymptote is at:

$$Mb^* = (Tb(S_1;\psi)_{min} + Z) \cdot \frac{1}{Txd(S;\chi_b)} \qquad (6.1.4b)$$

$$= N^* + Mb \qquad (6.1.4c)$$

The significance of Mb^* is like that of N^* for batch systems. Mb^* is that number of terminals which could be scheduled without interference. So for $M > Mb^*$, queueing is certain within the system.

Improvement of the bottleneck device uncovered by the above methods results in gains in system performance only until its value of $Txd(S;\chi_b)$ decreased to the

$$Txd(S;\chi_i) = \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

of some other subconfiguration in the system. As an example, consider a central subsystem for a terminal system having only a cpu and disk, and suppose the following data are collected:

$P(S;cpu) = 18\ Mw/s$ $\qquad$ $P(S;disk) = 100\ Kw/s$

$Wd(S;cpu) = 6\ Mw$ $\qquad$ $Wd(S;disk) = 50\ Kw$

Think time $Z = 15$ seconds

From the above data we have:

$Txd(S;cpu) = 1/3$ second $\qquad$ $Txd(S;disk) = 1/2$ second

So the bottleneck device is the disk. We plot the response time asymptotes in Figure 6.1.3a by first computing the minimum response time.

$$Tb(S_1;\psi)_{min} = \sum_{i=1}^{k} \frac{Wd(S;\chi_i)}{P(S;\chi_i)} = 1/3 + 1/2 = 5/6 \text{ second}$$

The response time asymptote for the bottleneck device (the disk) is given by:

$$Tb(S_1;\psi)_a = M \cdot Txd(S;disk) - Z = M \cdot \frac{1}{2} - 20$$

with $N^* = \frac{1/2 + 1/3}{1/2} = 1.67$

$$Mb = \frac{15}{1/2} = 30$$

$Mb^* = 30 + 1.67 \simeq 31.7$ terminals.

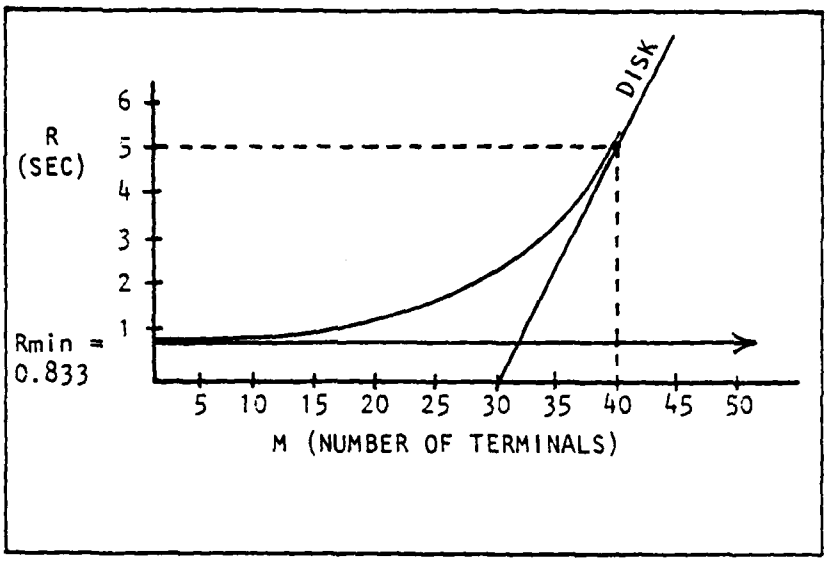


Figure 6.1.3a

Bottleneck Analysis Example

Disk Bottleneck Action

We should realize that the cpu is a potential system bottleneck, that is, were disk power sufficiently increased, the cpu would become the bottleneck. This more complex characterization is depicted in Figure 5.1.3b. Improvement obtained from a speedup of the disk is limited by the potential bottleneck action of the cpu.

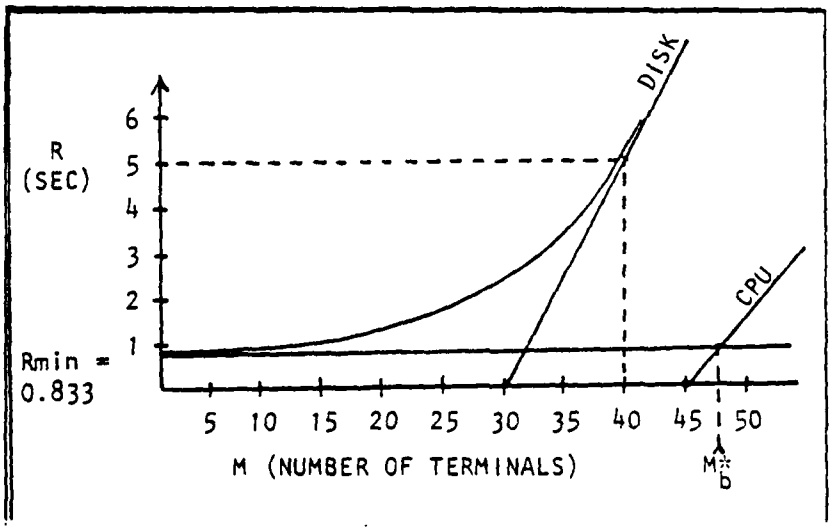


Figure 6.1.3b

Bottleneck Analysis Example

Potential Bottleneck Action of the CPU

6.2 STATE SPACE BALANCE AND THE PRODUCT FORM SOLUTION

Buzen and Denning show by operational methods in [BUZE77a] that the rates of transitions in and out of system states satisfy balance equations similar to:

$$\sum_{i,j} p(\underline{W}_{ij})(q_{ij} + q_{i0}q_{0j})\frac{P(S;\chi_i)}{Wb(s,\chi_i)} I_j$$

$$= \sum_{i,j} p(\underline{W}) \sum_i \frac{P(S,\chi_i)}{Wb(S;\chi_i)} \qquad (6.2.1)$$

For all $\underline{Ww}(S;\psi)$ and $i,j = 1,\cdots,k$

Where: $p(\underline{W}_{ij})$ is the proportion of $Tx(S;\psi)$ that the system is in the state defined by:

$$\underline{Ww}(S;\psi)_{ij} = [Ww(S;\chi_1),\cdots,Ww(S;\chi_i) + Wb(S;\chi_i),\cdots$$
$$Ww(S;\chi_j) - Wb(S;\chi_j),\cdots]$$

and where:

$$\underline{W}(S;\psi) = [Ww(S;\chi_1),\cdots,Ww(S;\chi_i),\cdots,Ww(S;\chi_j),\cdots]$$

I_i is an indicator function defined by:

$I_i = 0$ when $Ww(S;\chi_i) = 0$

$= 1$ when $Ww(S;\chi_i) > 0$

They further show that these equations have a solution of the form:

$$p(\underline{W}) = \frac{Y_1^{n_1} Y_2^{n_2} \cdots Y_k^{n_k}}{G} \qquad (6.2.2)$$

where: The Y_i depend only on workload and processor parameters.

The n_i are the numbers of requests at each χ_i given by

$$n_i = \frac{Ww(S;\chi_i)}{Wb(S;\chi_i)}$$

and G is a normalization constant given by:

$$G = \sum_{\underline{W}\omega(S;\psi)} \prod_{i=1}^{k} {}^{v}_{i}{}^{n_i}$$

By observing that from Equation (5.3.2)

$$p(\underline{W}_{ij}) = \frac{{}^{v}_{i}}{{}^{v}_{j}}\, p(\underline{W})$$

since $p(\underline{W}_{ij}) = Y_1^{n_1} \cdots Y_i^{n_i+1} Y_j^{n_j-1} \cdots$

substitution in the balance equations (6.2.1) shows that the product form of Equation (6.3.2) is valid if and only if equations like:

$$Y_j \frac{P(S;\chi_j)}{Wb(S;\chi_j)} = \sum_{i=1}^{k} Y_i \frac{P(S;\chi_i)}{Wb(S;\chi_i)}(c_{ij} + q_{i0}q_{0j}) \qquad (6.2.3a)$$

$$j = 1,\cdots,k$$

are valid.

Now if we substitute for Y_i the quantity

$$\frac{P(S;\chi_i,\psi)}{P(S;\chi_i)}$$ into Equation (6.2.3), we get:

$$\frac{P(S;\chi_j,\psi)}{Wb(S;\chi_j)} = \sum_{i=1}^{k} \frac{P(S;\chi_i,\psi)}{Wb(S;\chi_i)}(q_{ij} + q_{i0}q_{0j}) \qquad (6.2.3b)$$

If we add the additional equation:

$$\frac{P(S;\chi_0,\psi)}{Wb(S;\chi_0)} = \frac{P(S;\psi)}{Wd(S;\psi)} = \sum_{i=1}^{k} \frac{P(S;\chi_i,\psi)}{Wb(S;\chi_i)}\, q_{i0}$$

We then get

$$\frac{P(S;\chi_j,\psi)}{Wb(S;\chi_j)} = \sum_{i=0}^{k} \frac{P(S;\chi_i,\psi)}{Wb(S;\chi_i)} q_{ij}$$

$$= \frac{P(S;\psi)}{Wd(S;\psi)} q_{0j} + \sum_{i=1}^{k} \frac{P(S;\chi_i,\psi)}{Wd(S;\chi_i,\psi)}$$

and when each side is multiplied by

$$\frac{Wb(S;\chi_j)}{P(S;\psi)}$$

we get

$$D(S;\chi_j,\psi) = \frac{Wb(S;\chi_j)}{Wd(S;\psi)} q_{0j} + \sum_{i=1}^{k} D(S;\chi_i,\psi) \frac{Wb(S;\chi_j)}{Wb(S;\chi_i)} q_{ij} \quad (6.2.3c)$$

$$j = 0,\cdots,k$$

But these are just the earlier Equations (5.2.5), the Work Distribution Node Equations.

Since we also have under configuration work flow balance that

$$\frac{P(S;\chi_i,\psi)}{Wd(S;\chi_i)} = \frac{P(S;\chi_j,\psi)}{Wd(S;\chi_j)} \quad \text{all} \quad i,j = 0,\cdots,k$$

If

$$Y_i = \frac{P(S;\chi_i,\psi)}{P(S;\chi_i)} \quad \text{is a solution of the Equations (6.2.3a),}$$

then

$$Y_i = \frac{P(S;\chi_i,\psi)}{P(S;\chi_i)} \cdot \frac{Wd(S;\chi_i)}{P(S;\chi_i,\psi)} = \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

is also a solution.

Notice that since

$\frac{P(S;\chi_i,\psi)}{P(S;\chi_i)} = U(S;\chi_i,\psi)$, the utilization, then $\frac{Y_i}{Y_1}$ is the

relative utilization of χ_i to χ_1, a fact which we used in our previous bottleneck analysis.

We thus have a solution of products of terms

$$Y_i^{n_i} = \left[\frac{Wd(S;\chi_i)}{P(S;\chi_i)}\right]^{n_i}$$

and here the $Wd(S;\chi_i)$ are parameters of the workload and $P(S;\chi_i)$ are parameters of the subconfigurations. The n_i which define a system state depend on the work waiting at the subconfiguration χ_i and the blocksize work of χ_i, an implementation parameter.

6.3 ALGORITHMS FOR COMPUTING PERFORMANCE QUANTITIES

We now present two methodologies for the computation of system throughputs and device queue lengths and utilizations for systems having product form solutions. Each methodology has its distinct point of view and each is presented in a software physics formulation. The first is based on work by Reiser [REIS78], and is developed from intuitively appealing principles. The second, is an adaptation of Buzen's algorithm [BUZE73] applied to systems where the average work demand increases proportionately to the number of concurrent interactions in the system. We will present these for the case of single job classes only, although their bases have been extended elsewhere to multiple classes [REIS78,ROOD78].

6.3.1 Mean Value Analysis - Fixed Service/Workload Parameters

This approach to performance quantity computation depends on these principles:

(1) Upon arrival, an interaction "sees" a system as one with itself removed in long-term equilibrium.

(2) Little's Law is applied to the system as a whole and to individual queues.

The first principle is a consequence of the theorem stated by Reiser in [REIS78] and here paraphrased:

> "In a closed queueing network with product-form solution, the probability to see a state $\underline{N}(S;\psi)$ upon customer arrival when there are N interactions in a system is the same as the long term equilibrium probability of $\underline{N}(S;\psi)$ in the system with $N-1$ interactions present."

Letting $\hat{T}w(S_1;\chi_i)[N]$ be the waiting time required by a single visit to χ_i when the system load is N, we write:

$$\hat{T}w(S_1;\chi_i)[N] = \frac{Wb(S;\chi_i)}{P(S;\chi_i)} + \frac{\bar{W}w(S;\chi_i)[N-1]}{P(S;\chi_i)} \qquad (6.3.1)$$

That is, the waiting time consists of the time to complete $Wb(S;\chi_i)$ units of work from the arriving request plus the $\bar{W}w(S;\chi_i)[N-1]$ units of work waiting when the interaction arrives.

Now, the interaction S_1 will require

$\frac{Wd(S;\chi_i)}{Wb(S;\chi_i)}$ separate visits to χ_i, so we write for the accumulated waiting time:

$$Tw(S;\chi_i)[N] = \frac{Wd(S;\chi_i)}{Wb(S;\chi_i)} \cdot \hat{T}w(S;\chi_i)[N]$$

$$= \frac{Wd(S;\chi_i)}{P(S;\chi_i)}\left[1 + \frac{\bar{W}w(S;\chi_i)[N-1]}{Wb(S;\chi_i)}\right] \qquad (6.3.2)$$

This equation gives the accumulated waiting time in terms of the total work required at χ_i by the interaction S_1 over its entire life, the absolute power of χ_i and the average request count

$\frac{\bar{W}w(S;\chi_i)[N-1]}{Wb(S;\chi_i)}$ seen upon arrival.

We now sum the accumulated waiting time over all χ_i to get the total time that the interaction is in the system, either in queues or in service. This time is "busy" time at the configuration level so we write:

$$Tb(S_1;\chi)[N] = \sum_{i=1}^{k} Tx(S_1;\chi_i)[N] \qquad (6.3.3)$$

Note that we have made explicit use of our assumption that an interaction is present only at a single subconfiguration of a partition of ψ at any given instant.

We now apply Little's Law to the system having N similar transactions to get the throughput of work:

$$P(S;\psi)[N] = \frac{N \cdot Wd(S;\psi)}{Tb(S_1;\psi)[N]} \qquad (6.3.4a)$$

or for the interaction throughput:

$$X_0(S)[N] = \frac{P(S;\psi)[N]}{Wd(S;\psi)} = \frac{N}{Tb(S_1;\psi)[N]} \qquad (6.3.4b)$$

The average work waiting (including that in service) at a device is given by the application of Little's Law at that subconfiguration:

$$\bar{W}w(S;\chi_i)[N] = \hat{T}w(S_1;\chi_i)[N] \cdot P(S;\chi_i,\psi)[N]$$

First multiplying the right hand side by 1 in the form of

$$\frac{Wd(S;\chi_i)}{Wb(S;\chi_i)} \cdot \frac{Wb(S;\chi_i)}{Wd(S;\chi_i)}$$

we get:

$$\bar{W}w(S;\chi_i)[N] = Wb(S;\chi_i) \frac{P(S;\chi_i,\psi)[N]}{Wd(S;\chi_i)} \cdot Tw(S;\chi_i)[N]$$

and multiplying the right-hand side again by 1 in the form of

$\frac{Wd(S;\psi)}{Wd(S;\psi)}$ and recalling that

$P(S;\psi)\cdot\frac{Wd(S;\chi_i)}{Wd(S;\chi)} = P(S;\chi_i,\psi)$ we get:

$$\overline{Ww}(S;\chi_i)[N] = Wb(S;\chi_i)\frac{P(S;\psi)[N]}{Wd(S;\psi)}\cdot Tw(S;\chi_i)[N] \qquad (6.3.5a)$$

Since the recursion (6.3.1) is in terms of the average request count

$\frac{\overline{Ww}(S;\chi_i)}{Wz(S;\chi_i)}$ it is more convenient to write:

$$\frac{\overline{Ww}(S;\chi_i)[N]}{Wb(S;\chi_i)} = \frac{P(S;\psi)[N]}{Wd(S;\psi)}\cdot Tw(S;\chi_i)[N] \qquad (6.3.5b)$$

Or in terms of the interaction throughput:

$$\frac{\overline{Ww}(S;\chi_i)[N]}{Wb(S;\chi_i)} = X_0(S)[N]\cdot Tw(S;\chi_i)[N] \qquad (6.3.5c)$$

Finally we derive the utilization of χ_i, $U(S;\chi_i,\psi)$ directly from:

$$U(S;\chi_i,\psi) = \frac{P(S;\chi_i,\psi)[N]}{P(S;\chi_i)}$$

$$= \frac{P(S;\chi_i,\psi)[N]\cdot\frac{Wd(S;\psi)}{Wd(S;\chi_i)}}{P(S;\chi_i)\cdot\frac{Wd(S;\psi)}{Wd(S;\chi_i)}}$$

$$= \frac{P(S;\psi)[N]}{Wd(S;\psi)}\cdot\frac{Wd(S;\chi_i)}{P(S;\chi_i)} \qquad (6.3.6a)$$

Or in terms of interaction throughput:

$$U(S;\chi_i,\psi) = X_0(S)[N]\cdot\frac{Wd(S;\chi_i)}{P(S;\chi_i)} \qquad (6.3.6b)$$

6.3.1.1 *An Example - Mean Value Analysis*

We apply the "mean value" algorithms derived in the previous section to an example interactive system.

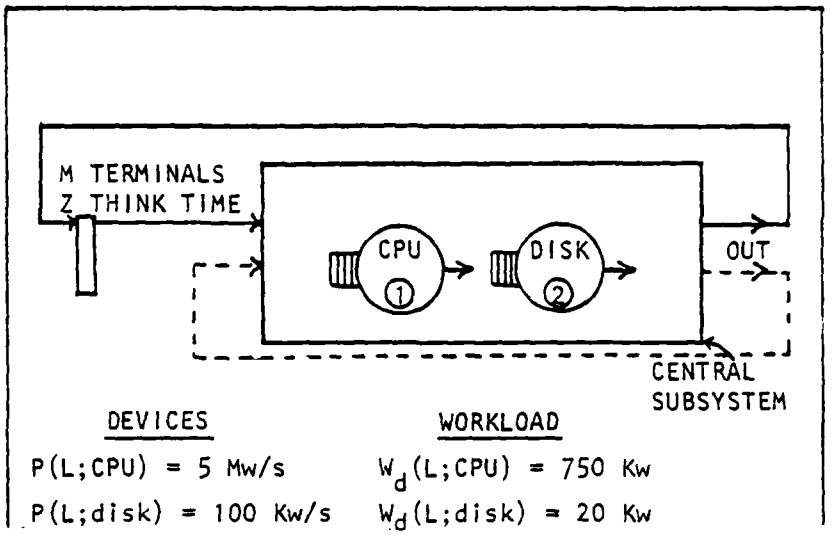


Figure 6.3.1

Example Interactive System

We will use our algorithms to solve the closed central subsystem, whose interaction flow is indicated by the dotted line in Figure 6.3.1. The analysis of this system together with its terminals will be considered in Section 6.4. First, let us sketch a solution using the bottleneck analysis techniques of the earlier part of this discussion.

From the provided data:

$$\frac{Wd(S;cpu)}{P(S;cpu)} = \frac{750 \times 10^3}{5 \times 10^6} = \underline{0.15 \text{ seconds}}$$

and

$$\frac{Wd(S;disk)}{P(S;disk)} = \frac{20 \times 10^3}{100 \times 10^3} = \underline{0.2 \text{ seconds}}$$

So the disk is the bottleneck device. The asymptotic system throughput (determined by the disk) is:

$$X_0(S)_{max} = 1/0.2 = 5 \text{ interactions/second}$$

The minimum (1 - interaction) response time is:

$$Tb(S_1;\psi)_{min} = 0.15 + 0.2 = 0.35 \text{ seconds} ,$$

so the 1 - interaction throughput is:

$$1/Tb(S_1;\psi)_{min} = 1/0.35 = 2.86 \text{ interactions/second}$$

Finally, the system saturation point is:

$$y^* = \frac{P(S;disk)}{Wd(S;disk)} \cdot \sum_i \frac{Wd(S;\chi_i)}{P(S;\chi_i)}$$

$$= 1/0.2 \quad (0.15 + 0.2) = 1.75$$

The resulting throughput function is sketched in Figure 6.3.2

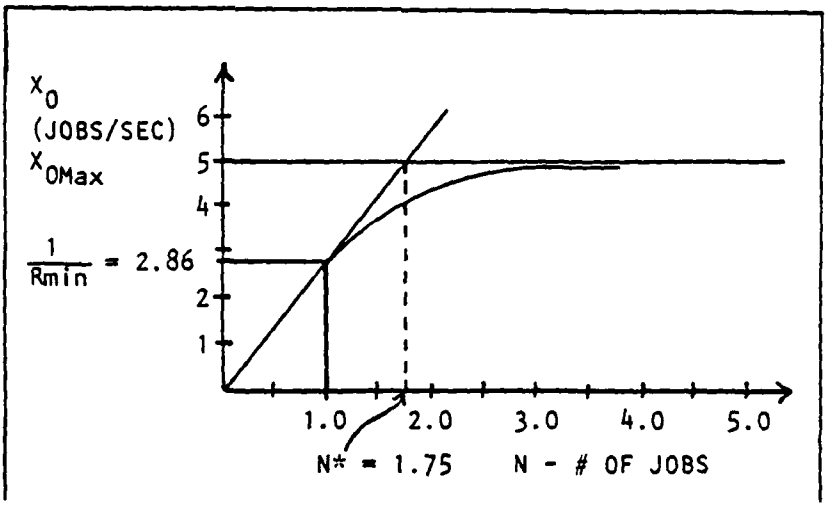


Figure 6.3.2

Central System Throughput - Bottleneck Analysis

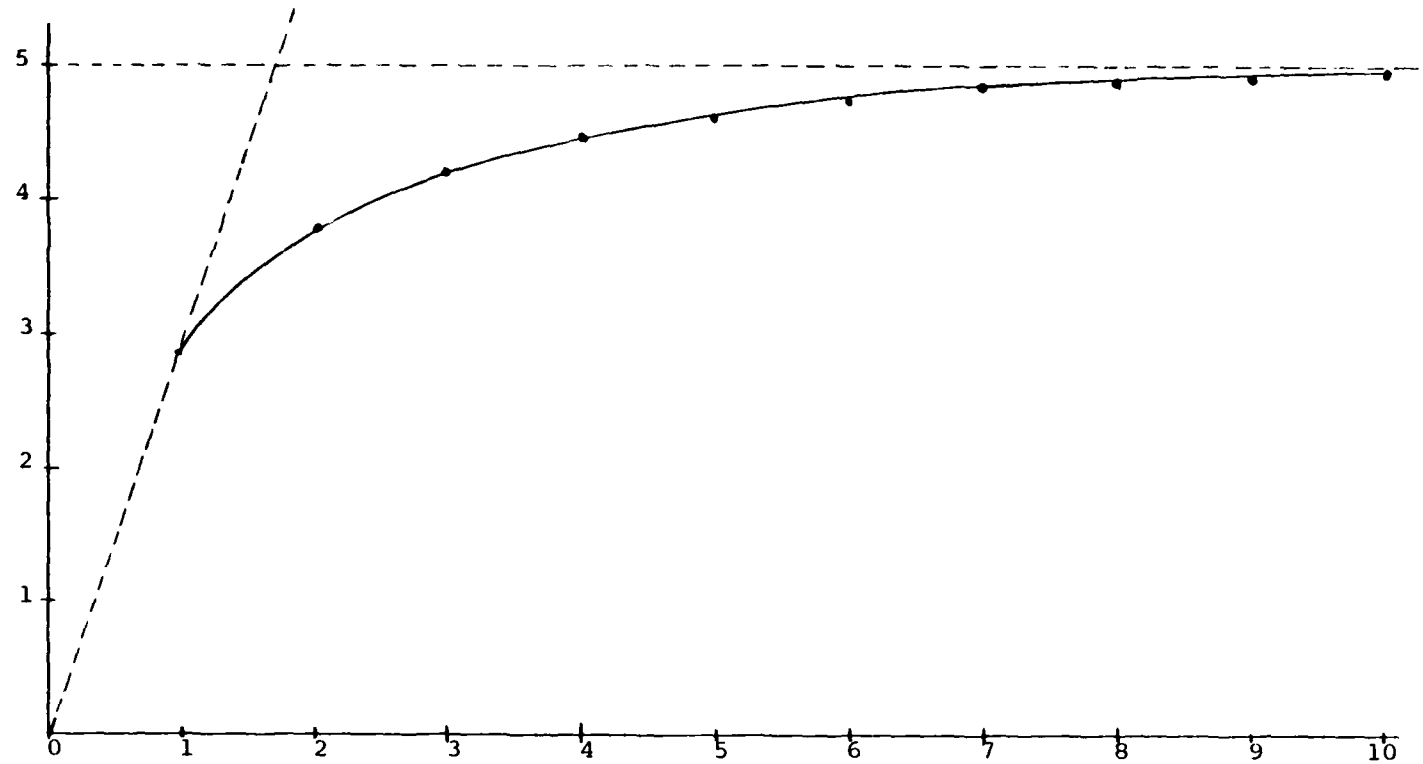
We now apply the algorithms of the previous section to compute the exact performance quantities. We start the recursion by setting $\bar{W}\omega(S;\chi_i)[0] = 0$. The process and results are summarized in Table 6.3.1. Note by observing the device utilization columns in the table that, as expected, it is the disk which shows the highest utilization at each level of multiprogramming load. The interaction throughput function $X_0(S)[N]$ is plotted in Figure 6.3.3 and realizes in exact form the approximating asymptotic sketch of Figure 6.3.2.

CPU	DISK	CONFIGURATION
$Wd(S;cpu) = 750Kw$	$Wd(S;disk) = 20Kw$	$Wd(S;\psi) = 770Kw$
$P(S;cpu) = 5Mw/s$	$P(S;disk) = 100Kw/s$	

N (Interactions)	$Tw(S_1;cpu)$ (sec)	$Tw(S_1;disk)$ (sec)	$Tb(S_1;\psi)$ (sec)	Xo(s) (Int/sec)	$P(S;\psi)$ (kw/s)	$\frac{\bar{W}w(S;cpu)}{\bar{W}b(S;cpu)}$	$\frac{\bar{W}w(S;disk)}{\bar{W}b(S;disk)}$	$U(S;cpu,\psi)$	$U(S;disk,\psi)$
1	0.150	0.200	0.350	2.857	2200	0.429	0.571	0.429	0.571
2	0.214	0.314	0.528	3.788	2917	0.811	1.19	0.568	0.578
3	0.272	0.438	0.710	4.23	3254	1.15	1.85	0.634	0.845
4	0.323	0.570	0.893	4.48	3449	1.45	2.55	0.672	0.896
5	0.368	0.710	1.08	4.64	3571	1.71	3.29	0.696	0.928
6	0.407	0.858	1.26	4.74	3654	1.93	4.07	0.712	0.949
7	0.439	1.01	1.45	4.82	3708	2.12	4.88	0.722	0.963
8	0.467	1.18	1.64	4.87	3746	2.27	5.73	0.730	0.973
9	0.491	1.35	1.84	4.90	3773	2.40	6.60	0.735	0.980
10	0.511	1.52	2.03	4.93	3793	2.52	7.48	0.739	0.985

TABLE 6.3.1 - MEAN VALUE ANALYSIS - EXAMPLE CENTRAL SUBSYSTEM

TABLE 6.3.1



CENTRAL SYSTEM THROUGHPUT - MEAN VALUE ANALYSIS

Figure 6.3.3

6.3.2 Buzen's Algorithm With Overhead Performance Degradation

In [BUZE73] Buzen demonstrates efficient algorithms for the computation of the normalizing constant, G, in the product form solution. In a subsequent paper [BUZE77] he shows how G computed at various levels of multiprogramming load denoted by $G[N]$ are sufficient in themselves for the computation of the performance quantities viz.,

$$X_0(S)[N] = \frac{G[N-1]}{G[N]} \tag{6.3.7}$$

$$U(S;\chi_i,\psi)[N] = \frac{Wd(S;\chi_i)}{P(S;\chi_i)} \cdot \frac{G[N-1]}{G[N]} \tag{6.3.8}$$

And the recursion for queue lengths:

$$\frac{\bar{W}w(S;\chi_i)[N]}{Wb(S;\chi_i)} = U(S;\chi_i,\psi)[N] \cdot 1 + \frac{\bar{W}w(S;\chi_i)[N-1]}{Wb(S;\chi_i)}$$

$$\text{in which } Ww(S;\chi_i)[0] = 0 \tag{6.3.9}$$

Buzen also shows a generalized version of the algorithm in [BUZE73] which admits the use of load dependent servers, i.e., load dependent ratios

$$\frac{Wd(S;\chi_i)}{P(S;\chi_i)} \quad \text{in our terminology.}$$

We first repeat the expression of the recursive algorithm here for the general term $g(n,m)$ in the recursion and then further develop a special case which has a simple form. First Buzen gives:

$$g(n,k) = \sum_{j=0}^{n} \frac{(Y_r)^j}{A_r(j)}\, g(n-j,k-1) \tag{6.3.10}$$

with $g(0,r) = 1$

and $A_r(j) = \begin{cases} 1 & \text{if } j = 0 \\ \prod_{i=1}^{j} a_r(i) & \text{if } j > 0 \end{cases}$

The $\alpha_r(i)$ are the arbitrary load dependent factors which divide the

$$Y_r = \frac{Wd(S;\chi_r)}{P(S;\chi_r)}$$

to express the variation of service time at each load i.

Now suppose that for our special case we have that:

$$Wd(S;\chi_i)[N] = (1+D)Wd(S;\chi_i)[N-1] \qquad (6.3.11)$$

$$N = 2,3,\cdots$$

where D is a constant factor of increase in interaction work demand for each increment of multiprogramming load over one.

Defining the interaction demand execution time:

$$Txd(S;\chi_i)[N] \equiv \frac{Wd(S;\chi_i)[N]}{P(S;\chi_i)}$$

and letting $Txd(S;\chi_i)[1]$ be more simply denoted as $Txd(S;\chi_i)$, we have:

$$Txd(S;\chi_i)[n] = Txd(S;\chi_i)\cdot(1+D)^{n-1}$$

Now the m^{th} term in the sum in Equation (6.3.10) is:

$$[Txd(S;\chi_i)]^m\cdot(1+D)^{m-1}\cdot(1+D)^{m-2}\cdots(1+D)^{m-m}$$

which we may rewrite as:

$$[Txd(S;\chi_i)]^m(1+D)^{S(m-1)} \text{ where } S(k) = \sum_{i=1}^{k} i$$

Figure 6.3.4 conceptualizes the recursion in a two dimensional array. Each column represents a computation with parameters for a different device. The zero$^{\text{th}}$ element in each column contains a "one." There is a zero$^{\text{th}}$ column with entries of zero for loads 1 through N to start the recursion. One proceeds by computing $g(n,r)$ by progressing down a column beginning with $r=1$. The terms summed as indicated give $g(n,r)$ and the final column has the elements that correspond to the elements $G[N]$ of Equations (6.3.7, 6.3.8, 6.3.9).

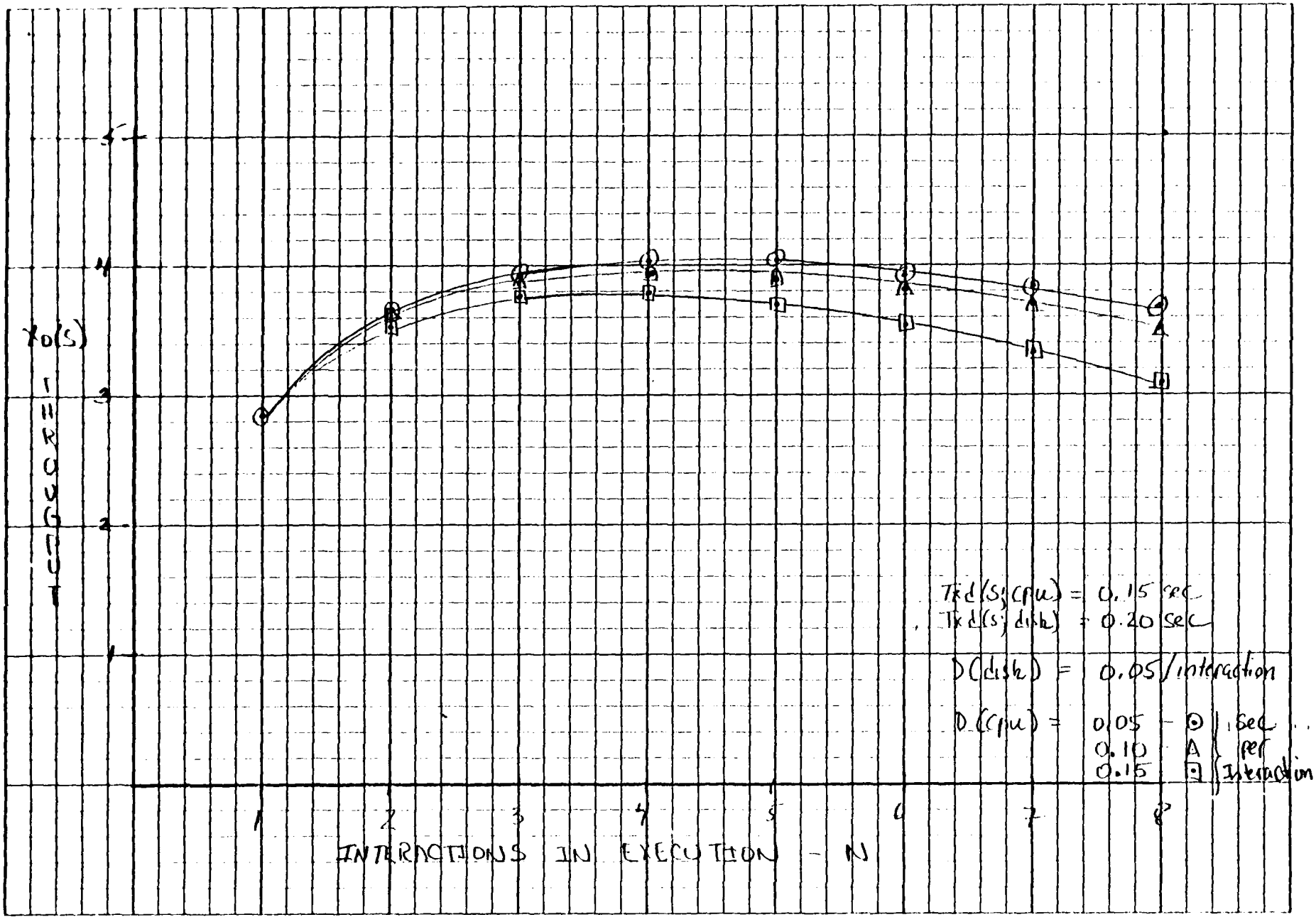
DEVICES

		0	1	2	•	•	•	•	•	k	•	•	•	•	K
	0	1	1	1						$1 \cdot (Txd)^{n} \ (1 + D)^{S(n-1)}$	+				1
	1	0								$g(1, k - 1) \cdot (Txd)^{n-1} \ (1 + D)^{S(n-2)}$	+				
L	2	0								$g(2, k - 1) \cdot (Txd)^{n-2} \ (1 + D)^{S(n-3)}$	+				
O	•	•								•					
A	•	•								•					
D	n-2	•								$g(n - 2, k - 1) \cdot (Txd)^{2} \ (1 + D)$	+				
S	n-1	•								$g(n - 1, k - 1) \cdot (Txd)^{1}$	+				
	n	0								$g(n, k - 1) \cdot (Txd)^{0}$	+	→ $g(n,k)$			
	•	•													
	•	•													
	•	•													
	N	0													

$$S(n) = \sum_{i=1}^{n} i$$

BUZEN'S ALGORITHM - LOAD DEPENDENT (PROPORTIONAL DEGRADATION)

Figure 6.3.4



Example System - Load Dependent Throughputs
Figure 6.3.5

6.3.2.1 *The Example System - Continuation*

We can now incorporate overhead work demand increases according to the scheme just developed into the system of Section 6.3.1.1. Recall:

$$Txd(S;cpu) = \frac{Wd(S;cpu)}{P(S;cpu)} = 0.15 \text{ seconds}$$

$$Txd(S;disk) = \frac{Wd(S;disk)}{P(S;disk)} = 0.2 \text{ seconds}$$

For the disk we take a constant 5% increase in overhead work for each increment in load. For the cpu we take possible increases to be 5%, 10%, and 15% and examine each case. The resulting predicted throughputs are given in Table 6.3.2. These are plotted in Figure 6.3.5 and show what is usual for systems with multiprogramming load - increasing overheads, the throughput function achieves a maximum and then starts a decline for further increases in load.

6.3.2.2 *A Graphical Decomposition Solution*

We now return to our example system as originally given; a collection of terminals interacting with the central subsystem just analyzed. Courtois [COUR75] has observed that systems are *nearly completely decomposable* into groups of smaller subsystems if the state changes between the subsystems occur at a rate much slower than those within the subsystems. Thus, observing that the rate of interaction submission from the terminals for M signed-on terminals is:

$\frac{M - N}{Z}$ where N is the number of interactions in process in the central subsystem,

and assuming that the maximum submission rate, given by M/Z, is much less than the rate of state transitions within the central subsystem, we can treat the composite as the "joining" of two equivalent components each with characteristics separately determined as a function of the load N. A useful graphical

CPU

Wd(S;cpu) = 750 Kw/s

P(S;cpu) = 5 Mw/s

DISK

Wd(S;disk) = 20 Kw

P(S;disk) = 100 Kw/s

DEGR(disk) = 0.05/Interaction

LOAD (Interactions)	THROUGHPUT (Interactions/Sec) DEGR(CPU)/Interaction		
	0.05	0.10	0.15
1	2.858	2.858	2.858
2	3.660	3.618	3.576
3	3.954	3.871	3.785
4	4.046	3.929	3.797
5	4.035	3.894	3.710
6	3.965	3.806	3.557
7	3.859	3.687	3.351
8	3.729	3.546	3.091
9	3.584	3.389	2.778
10	3.429	3.219	2.419

Table 6.3.2

Example System - Load Dependent Throughputs

method for this joining process is conceptually shown in Figure 6.3.6. A typical central subsystem throughput function is plotted and its intersections with the terminal subsystem interaction submission function

$\frac{M - N}{Z}$ are indicated by Na, Nb and Nc.

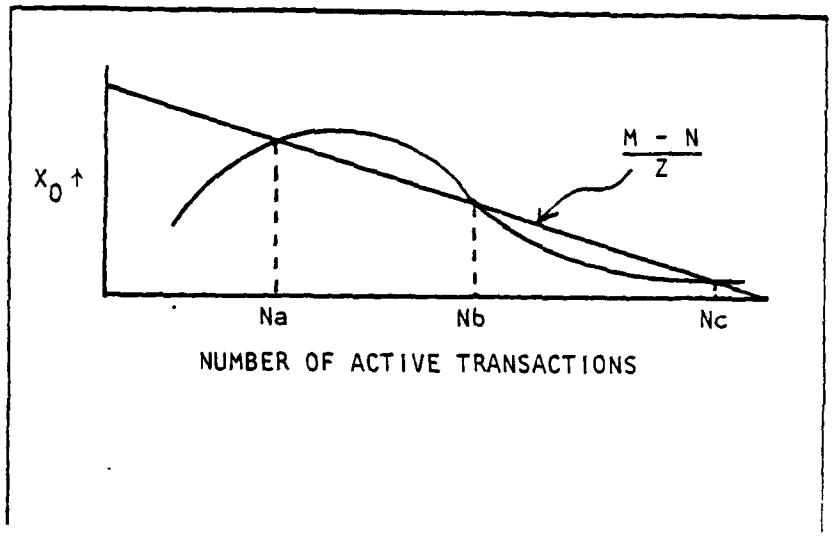


Figure 6.3.6

Decomposition Analysis - Conceptual

Each of these points is one where interaction submission rate equals the interaction throughput (completion rate) of the central subsystem. However, of the three solutions only Na and N_c are *stable*, that is, these points have the property of attracting the subsystem from neighboring values of N. The point Nb, though a solution, does not have this property. To show this, first consider a stable point such as Na. If the number of interactions in the central subsystem should increase to a value $Na + \Delta n$, the central subsystem responds with an increased throughput which exceeds the submission rate and thus tends to return the subsystem interaction count to Na. Similarly, a

decrease to $Na - \Delta n$ is met with a lowering of throughput below the submission rate. This results in increased congestion raising the subsystem interaction count back towards Na. At Nb, however, an increase to $Nb + \Delta n$ is met with a <u>decrease</u> of throughput thus causing further and further congestion until stability is reached at Nc. A decrease to $Nb - \Delta n$ is responded to with <u>increased</u> throughput, driving the subsystem towards Na.

Now we would like to stablize the systems at a point like Na to the left of the maximal throughput in the interest of minimizing the response time observed by a terminal user. This time, given by an application of Little's Law, is:

$$Tb(S;\psi) = \frac{Na}{X_0(S)}$$

where $X_0(S)$ is the throughput at Na.

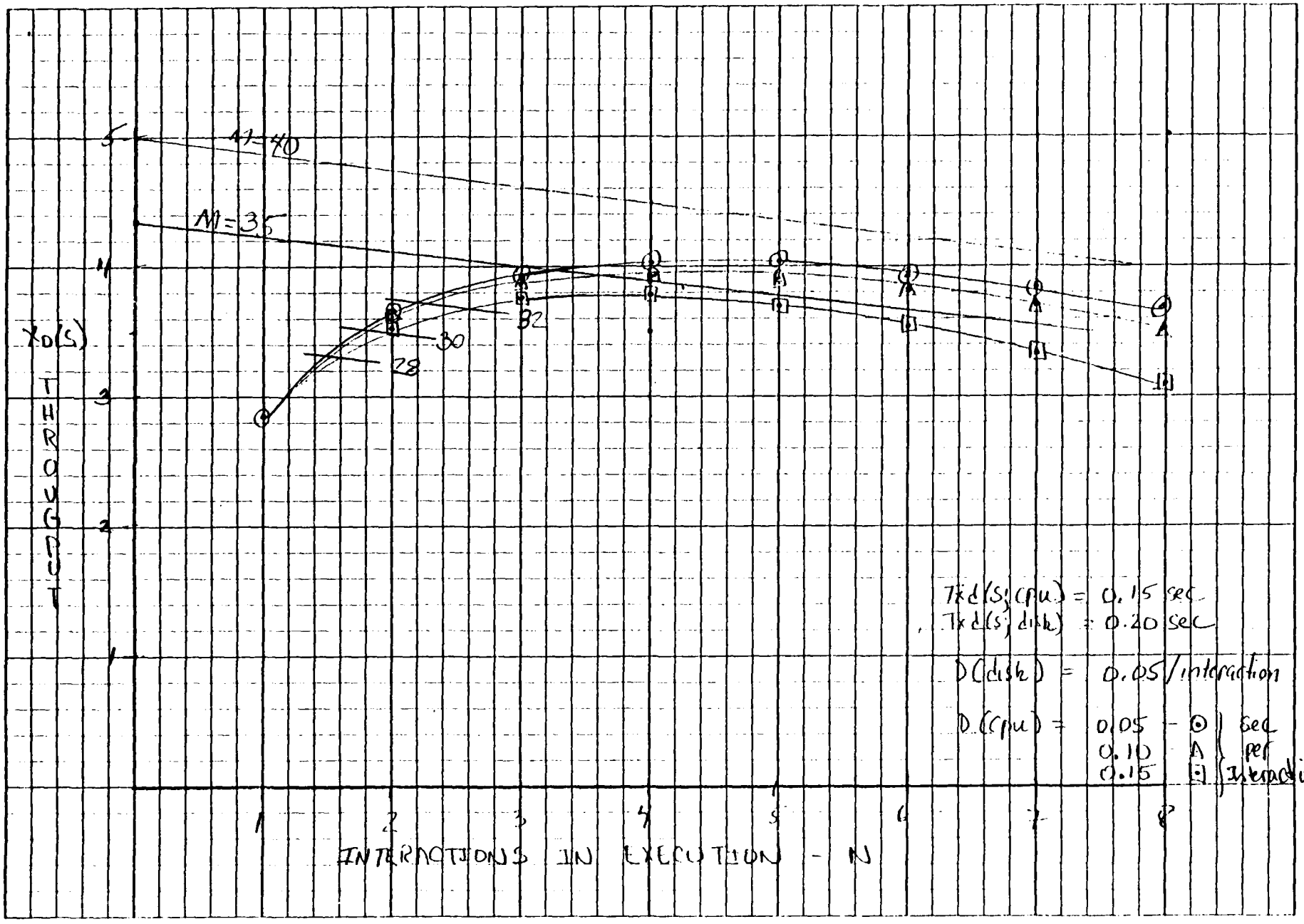
The effects of varying the number of signed-on terminals, M, in our example can now be observed by indicating with lines or tick marks, the intersection of the lines

$\frac{M - N}{Z}$ with the central subsystem throughput functions.

This is done in Figure 6.3.7 and we make several observations based on the number of terminals signed-on and the cpu degradation factor which is operative. Specifically for this example:

(i) For 40 terminals, there is no intersection, Na, with any of the throughput functions such that the intersection is to the <u>left</u> of the peak throughput value.

(ii) For 35 terminals there are such intersections only for the cases where the cpu degradation is 0.05 or 0.10. If we were uncertain that the cpu degradation was within these limits, we should consider the choice of this number of terminals to be served to be at risk of not providing acceptable response time.



Example System - Graphical Decomposition
Figure 6.3.7

(iii) For the other values of M shown (28, 30, 32) we have all three cpu degradation factors providing solution points, Na, to the left of the maximal throughputs. So we are at the least risk of ending up stabilized at a solution point like Nc of the conceptual discussion corresponding to a low throughput and relatively large number of interactions in the system.

The response times corresponding to the solution points to the left of maximum throughput are given by:

$$Tb(S_i;\psi) = \frac{Na}{X_0(S)}$$

and are summarized in Table 6.3.3 below.

M (Terminals)	DEGR(cpu) = 0.05			DEGR(cpu) = 0.1			DEGR(cpu) = 0.15		
	$X_0(S)$	Na	$Tb(S_1;\psi)$	$X_0(S)$	Na	$Tb(S_1;\psi)$	$X_0(S)$	Na	$Tb(S_1;\psi)$
28	3.5	1.45	0.44	3.3	1.55	0.47	3.3	1.6	0.48
30	3.55	1.8	0.51	3.55	1.8	0.51	3.5	2.0	0.57
32	3.75	2.2	0.59	3.75	2.2	0.59	3.65	2.6	0.71
35	3.95	3.4	0.86	3.9	3.6	0.92	-	-	-

Table 6.3.3

REFERENCES

[BUZE73] Buzen, J. P., *"Computational Algorithms for Closed Queueing Networks with Exponential Servers"*, Communications of the ACM, Vol. 16, No. 9 (Sept. 1973), pp. 527-531.

[BUZE77a] Buzen, J. P. and Denning, P. J., *"Operational Analysis of Queueing Networks"*, published by the authors (1977).

[BUZE77b] Buzen, J. P., *"Operational Analysis: An Alternative to Stochastic Modelling"*, BGS Systems, Inc. Report, Lincoln, Mass. (Nov. 1977).

[COUR75] Courtois, P. J., *"Decomposability, Instabilities and Saturation in Multiprogramming Systems"*, Communications of the ACM, Vol. 18, No. 7 (July 1975), pp. 371-376.

[DENN78] Denning, P. J., and Buzen, J. P., *"The Operational Analysis of Queueing Network Models"*, Computing Surveys, Vol. 10, No. 3 (Sept., 1978) pp. 225-261.

[KLEI76] Kleinrock, L., Queueing Systems, Vol. II, Wiley (1976).

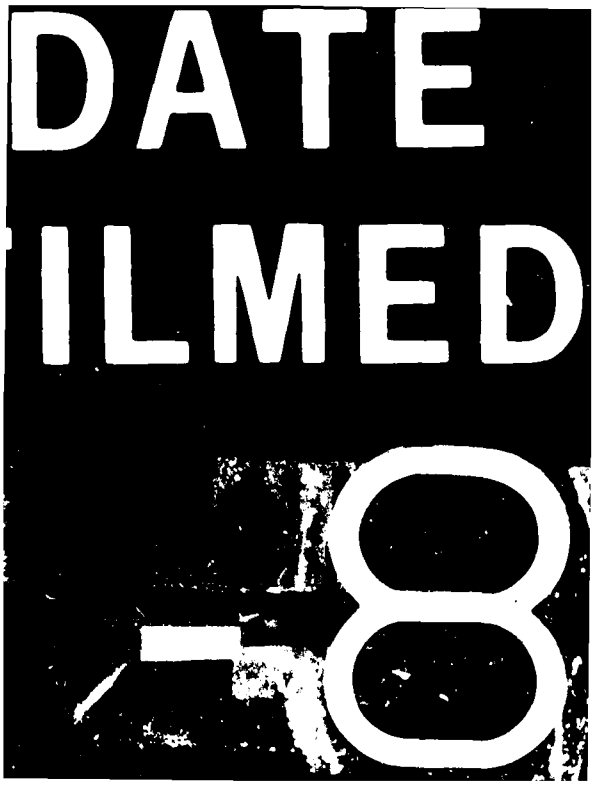
[KOLE76] Kolence, K. W., An Introduction to Software Physics, Institute for Software Engineering, Palo Alto, CA (1976).

[KOVA79] Kovach, R. P., *"Foundations and Concepts of Software Physics"*, Institute for Software Engineering, Palo Alto, CA (to appear).

[NEWE71] Newell, G. F., *"Applications of Queueing Theory"*, Chapman & Hall, Ltd. (London), 1971.

[REIS79] Reiser, M., *"Mean Value Analysis of Queueing Networks"*, IBM Research Report RC 72-28, Yorktown Heights (1978).

[ROOD78] Roode, J. D., *"Multiclass Operational Analysis of Queueing Networks"*, Technical Report, National Research Institute for Mathematical Sciences, (TWISK 56), Pretoria (Nov. 1978).


DATE
ILMED
-8